Coloration des graphes

Cours	Site
Coloration_Graphe.pdf	Arel

Liens	Fonction
https://www.youtube.com/watch?v=CUe7LC3CdH8	Coloration des graphes : application à la création de plannings
https://www.youtube.com/watch?v=w3LFOAw-J1I	Coloration des graphes 2ème partie
https://www.youtube.com/watch?v=aVngtCbleag	Cours sur la coloration des graphes
https://www.youtube.com/watch?v=JtvJr2lY-mc	Exercices
https://www.youtube.com/watch?v=XaFQozMVZKw	Algorithme de Welsh-Powell

Exercice 1

Coloration de carte

On considère la carte de la France suivante. On veut colorier chaque région de la France de telle sorte que deux régions voisines ne soient pas de la même couleur. Combien de couleur, au minimum, sont nécessaires pour colorier cette carte?

Aide : Réaliser un graphe en reliant par des arêtes les régions frontalières. Et appliquer l'algorithme de Welsh-Powell.
En déduire que le nombre minimum de couleurs nécessaires est égal à 4

Exercice 2

Organisation d'examens

Une école d'ingénieurs doit organiser les examens des enseignements optionnels de ses élèves de troisième année. Les différentes options sont : français (F) ; anglais (A) ; mécanique (M) ; sport (S) ; Internet (I) et dessin industriel (D). Certains étudiants ont choisi plusieurs options, et les regroupements existants sont : (F,A,M) ; (D,S) ; (I,S) ; (I,M). Combien de demi-journées seront-elles nécessaires à cette organisation sachant que la durée de chaque épreuve est d'une demi-journée ?

Exercice 3

Fréquences radio

On désire attribuer des canaux de fréquences radio à six stations. Deux stations distantes de moins de 150 km ne peuvent pas avoir le même canal. Combien faut-il de canaux distincts (au minimum) connaissant les données du tableau ci-dessous, exprimant la distance entre les stations ?

	A	B	C	D	E	F
A	–	85	175	200	50	100
B	85	–	125	175	100	160
C	175	125	–	100	200	250
D	200	175	100	–	210	220
E	50	100	200	210	–	100
F	100	160	250	220	100	–

Exercice 4

Bibliothèque

Sept élèves, désignés par A, B, C, D, E, F et G se sont rendus à la bibliothèque le même jour. Le tableau suivant précise "qui a rencontré qui" (la bibliothèque étant petite, deux élèves présents au même moment se rencontrent nécessairement...).

Élève	A	B	C	D	E	F	G
A rencontré	D, E	D, E, F, G	E, G	A, B, E	A, B, C, D, F, G	B, E, G	B, C, E, F

De combien de places assises (au minimum) doit disposer la bibliothèque pour que chacun ait pu travailler correctement au cours de cette journée?

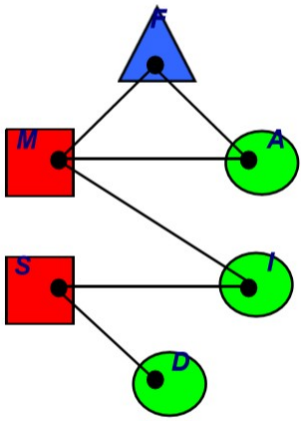
Exercice 5

Nombre chromatique

Écrire l'algorithme de Welsh-Powell qui permet de majorer le nombre chromatique d'un graphe.

Corrections

Exercice 2



Sommet	M	A	F	I	S	D
n° couleur	1	2	3	2	1	2

L'algorithme donne trois couleurs. Le graphe contient un graphe complet d'ordre 3 : F A M ; on ne pourra pas obtenir moins de trois couleurs. Le nombre chromatique de ce graphe est donc 3.

Exercice 4

On construit d'abord le graphe des rencontres : les sommets représentent les élèves; une arête (i, j) signale que les élèves i et j se sont rencontrés. Il reste alors à proposer une coloration du graphe

utilisant un nombre minimum de couleurs. Chaque couleur correspondra à une place assise. La coloration montre que la bibliothèque dispose d'au moins quatre places assises, car le graphe contient une clique à quatre sommets (B-E-F-G). Ces quatre places assises sont suffisantes.

Exercice 5

Algorithm : Algorithme de Welsh-Powell pour colorier un graphe

Data: Un graphe $G = (S, A)$
Result: Une coloration $\varphi : S \to \mathbb{N}$ de G

```
L ← liste des sommets ordonnés par degré décroissant ;
couleur-courante ← 0;
while L ≠ ∅ do
    couleur-courante ← couleur-courante +1;
    Colorier s le premier sommet de L avec couleur-courante;
    Eliminer s de L;
    V ← voisins de s;
    for x ∈ L do
        if x ∉ V then
            Colorier x avec la couleur-courante;
            Eliminer x de L;
            Ajouter les voisins de x à V;
```

De manière grossière, on passe |S| fois dans la boucle **Tant Que** puis |S| fois dans la boucle **POUR**, on a donc une complexité grossière en $O(|S|^2)$. Cependant ,on voit que l'on passe au maximum Δ(G) +1 fois dans la boucle **Tant Que**. On a donc une complexité en $O(\Delta(G)|S|)$.

Tournées eulériennes et hamiltoniennes

Cours	Site
Euler_Hamilton.pdf	Arel

Rapells de cours

Cycle simple : Cycle qui n'utilise pas deux fois la même arête.

Cycle eulériens : Cycle simple qui passe par toutes les arêtes d'un graphe non orienté.

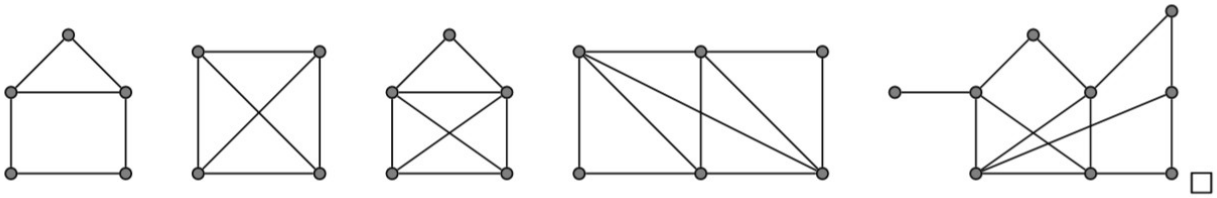
Cycle (respectivement une chaîne) hamiltonien Soit G un graphe non orienté. Un cycle (respectivement une chaîne) hamiltonien est un cycle (resp. une chaîne) qui passe une et une seule fois par tous les sommets de G. On définit lesmêmes notions pour un graphe orienté G: un circuit ou un chemin hamiltonien est un circuit ou un chemin passant une et une seule fois par tous les sommets de G

Proposition : *Un graphe dont tous les sommets sont de degré supérieur ou égal à 2 possède un cycle. En particulier, un graphe acyclique admet un sommet de degré 0 ou 1.*

Liens	Fonction
https://www.youtube.com/watch?v=DH0Hxes2nOo	Cours sur les parcours eulériens
https://www.youtube.com/watch?v=bm-VqHOX5vM	Graphe planaire
https://www.youtube.com/watch?v=uwJ27Yr7rZM	Cycle Hamiltonien
https://www.youtube.com/watch?v=8YwJl4vU_NA	Cas pratique : Trouver un chemin hamiltonien dans un tournoi

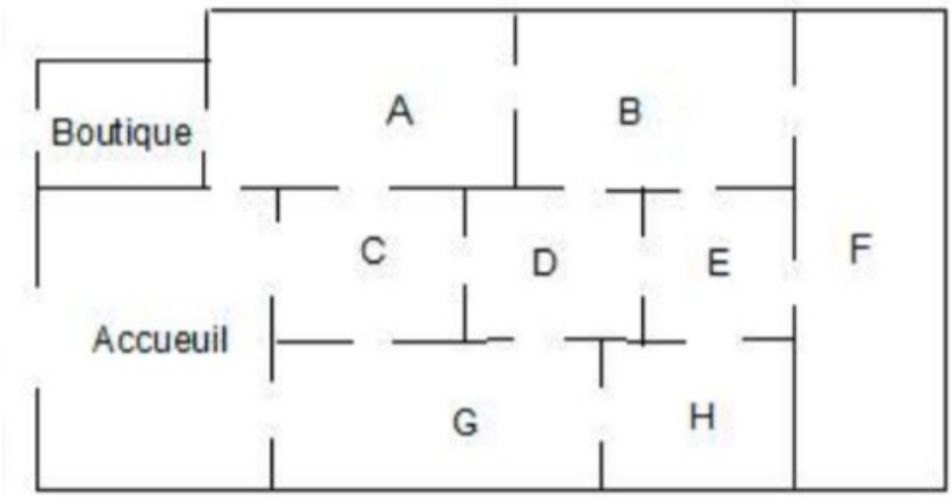
Exercice 1

Peut-on dessiner sans lever le crayon et en ne passant qu'une seule fois sur chaque arête les graphes ci-dessous ? Et revenir sur la position initiale ? Justifier.



Exercice 2

Voici le plan d'un musée : les visiteurs partent de l'accueil (Y),visitent le musée et doivent terminer leur visite à la boutique (Z).



Question 1

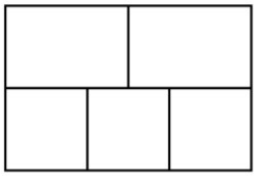
Représenter la situation à l'aide d'un graphe en précisant ce que représentent arêtes et sommets.

Question 2

Est-il possible de trouver un circuit où les visiteurs passent une fois et une seule par toutes les portes,entrant par l'accueil et ressortant par la boutique ?Justifier.Si oui,donner un exemple d'un tel circuit.

Exercice 3

Cinq pays sont représentés avec leurs frontières. Est-il possible de partir d'un pays et d'y revenir en franchissant chaque frontière une fois et une seule ? Justifier. Si oui, donner un exemple d'un tel circuit.



Exercice 4

On considère la carte de la France suivante. Est-il possible de trouver un trajet reliant toute les régions en ne traversant les "frontières" de chaque région qu'une et une seule fois ? Est-il possible de revenir à la région de départ ? Justifier. Si oui, donner un exemple d'un tel trajet.

Exercice 5

Préconditions d'Euler

Quelle(s) condition(s) doi(ven)t être satisfaite(s) pour qu'un graphe contienne un cycle eulérien ?

Écrire la méthode qui permet de tester si un graphe est eulérien.

Exercice 6

Soit G=(V,E) avec V l'ensemble des nœuds et E l'ensemble des arêtes.
Montrer que G contient un circuit Eulérien si et seulement si $d^-(v)=d^+(v)$, en notant par $d^-(v)$le degré entrant du nœud v et par $d^+(v)$, le degré sortant du nœud v.

Exercice 7

Cycle eulérien

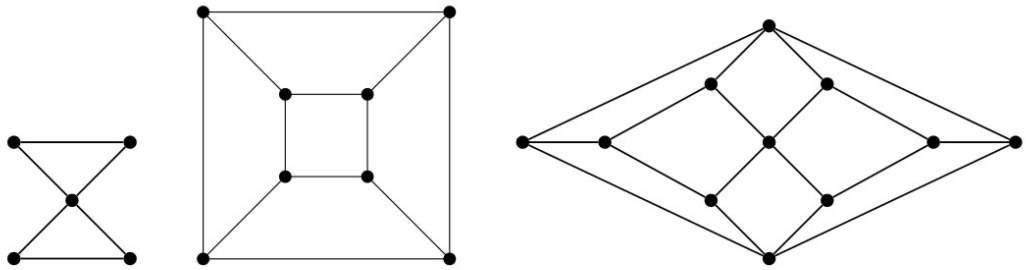
Écrire la méthode qui, sur un graphe eulérien, renvoit un cycle eulérien.

Rappel de l'algorithme :

1. *Construire un cycle quelconque C pour le graphe G*
2. *Si toutes les arêtes ne sont pas dans le cycle, alors*
3. *Déterminer les composantes connexes H_k de G privé des arêtes de C*
4. *Pour chaque composante connexe H_k*
   (a) *Trouver un cycle eulérien E_k*
   (b) *Intercaler le cycle eulérien E_k dans le cycle quelconque C*
5. *Le cycle C est eulérien pour le graphe G*

Exercice 8

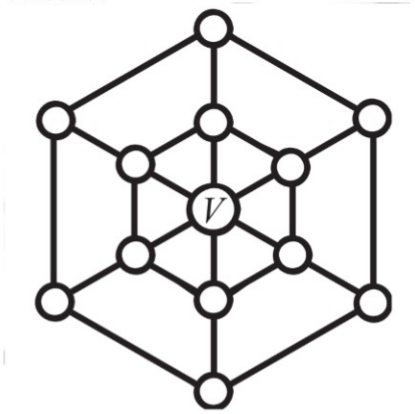
Peut-on trouver une chaîne/cycle qui ne passe qu'une seule fois sur chaque sommet des graphes ci-dessous?



Exercice 9

Le problème du voyageur de commerce

Étant données 13 villes reliées par des routes, un voyageur de commerce habitant la ville V peut-il passer par chaque ville une fois et une seule, en rentrant chez lui à la fin de son circuit?

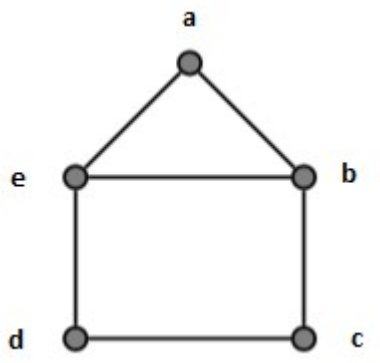


Note : *ce problème est connu sous le nom du problème du voyageur de commerce. Et aujourd'hui encore, on ne connaît pas sa solution générale.*

Exercice 10
Écrire la méthode qui permet de tester si un graphe est hamiltonien.

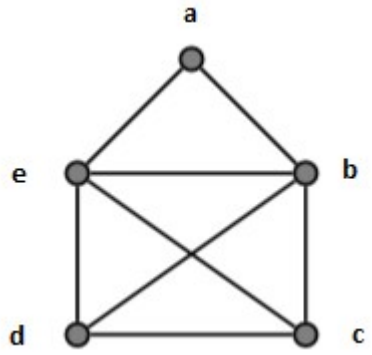
Corrections

Exercice 1
a)



Par exemple, considérons le chemin e-b-c-d-e-a-b.

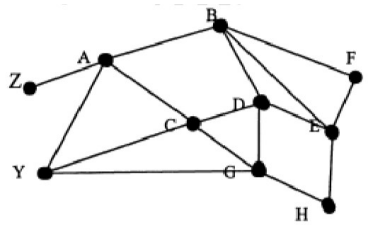
C)



Par exemple, considérons le chemin d-e-b-a-e-c-b-d-c

Exercice 2

Question 1

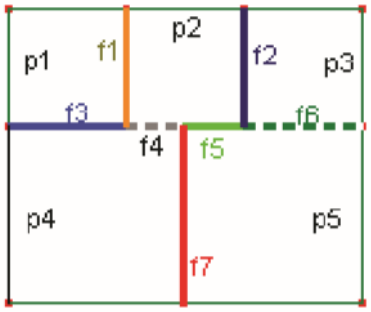


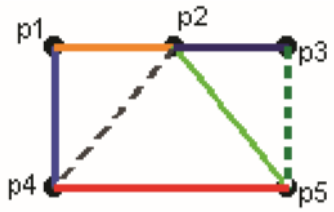
Les sommets représentent les salles de musées, les arêtes étant les portes. On schématise la boutique le nœud Z et l'accueil par le nœud Y.

Question 2

Trouver un tel circuit revient à trouver une chaîne eulérienne parcourant ce graphe. D'après la partie 1, une telle chaîne existe. Un exemple de tel circuit est la chaîne Y(accueil)-G-C-Y-A-C-D-G-H-E-D-B-E-F-B-A-Z(boutique), qui parcourt une et une seule fois toutes les arêtes du graphe.

Exercice 3





Oui, par application du théorème d'Euler.

Exercice 5

Algorithme de Fleury :
Entrée : *Un graphe non-orienté connexe G dont tous les sommets sont de degré pair.*
Sortie *: Un cycle eulérien C de G.*
Etape 1 :
Initialisation*. Choisir un sommet* u_0 *de* $G, C_0 := u_0, G_0 := G, i := 0$.
Etape 2 *:*
Construction du cycle.
Tant que u_i *n'est pas un sommet isolé faire :*
Choisir une arête $e_{i+1} = u_i u_{i+1}$ *de Gi incidente à* u_i*, si possible contenue dans un cycle de* G_i.
$C_{i+1} := C_i e_{i+1} u_{i+1}$,
$G_{i+1} := Gi - e_{i+1}$,
$i := i + 1$.
Etape 3 :
Fin de l'algorithme.
$C := Ci$,

STOP.

Avec C_i,G_i et u_i la suite, le graphe et le sommet de la $i^{-ème}$ itération . (a)
Les ensembles d'arêtes de C_i et de G_i forment une bipartition de l'ensemble des arêtes de G.
C_i est une chaîne simple d'extrémités u_0 et u_i.
G_i contient une chaîne eulérienne d'extrémités u_0 et u_i.
Une suite C construite par l'algorithme de Fleury est un cycle eulérien de G.

<u>Exercice 10</u>

La recherche de cycle hamiltonien est un problème connu pour être NP-complet. Plus simplement cela veut dire que l'on ne sait pas à l'heure actuelle s'il existe un algorithme polynomial qui teste si le graphe est hamiltonien. Une façon simple (mais pas la plus efficace!) de trouver un cycle hamiltonien consiste à examiner toutes les permutations σ des sommets, et à tester ensuite si la permutation σ correspond à un cycle. Or il existe n! permutations possibles sur n sommets. Donc en utilisant la célèbre formule de Stirling :

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n (1 + \Theta\left(\frac{1}{n}\right))$$

et en utilisant le log et l'exponentielle à bon escient nous obtenons $n! = O(e^{nlogn})$. Ce qui n'est pas polynomial!

Ford Fulkerkson

Cours	Site
Reseaux_de_transports_cours.pdf	Arel

Rappel de cours

Définition 1

Soit G=(V,E), avec V l'ensenmble de nœuds du graphe G et E, l'ensemble des arêtes du graphe G, un réseau de transport. Un flot circulant dans le réseau G est une fonction f définie sur l'ensemble des arcs E vérifiant la loi de conservation suivante

$$\forall v \neq s,t, \sum_{\{u \in V \text{tel que}(u,v) \in E\}} f(u,v) = \sum_{\{u \in V \text{ tel que}(v,u) \in E\}} f(v,u).$$

Définition 2

Soit G=(V,E), un réseau de transport dont on note c les capacités des arcs. On dit qu'un flot *f* circulant dans G est compatible si sa valeur sur chacun des arcs est inférieure à la capacité de l'arc en question, c'est-à-dire si $\forall (u,v) \in E, f(u,v) \leq c(u,v).$

Définition 3

Soit G=(V,E),un réseau de transport. On dit qu'un flot *f* circulant dans G est complet s'il est compatible et si chaque chemin de la source vers le puits possède un arc dont la valeur du flot est égale à sa capacité. Un tel arc est dit saturé.

Définition 4

Soit G=(V,E), un réseau de transport. On dit qu'un flot *f* circulant dans *G* est maximal s'il est compatible et s'il possède la plus forte valeur du flot parmi tous les flots compatibles.

Définition 5

Soit G=(V,E), un réseau de transport dont on note c les capacités des arcs. On considère un flot compatible f circulant dans G. Le graphe d'écart de ce flot f dans G est un graphe possédant les mêmes sommets que G, et dont les arcs vont dépendre de ceux de G ainsi que de la valeur du flot y circulant.

Pour un arc (u,v)de E, on construit un ou deux arcs dans le graphe d'écart selon la règle :

- Si $f(u,v)<c(u,v)$, on définit un arc (u,v) de valuation $c(u,v)-f(u,v)$.
- Si $f(u,v)>0$, on définit un arc (v,u) de valuation f(u,v).

De ce fait, pour un arc (u,v) dont la capacité n'est pas encore atteinte, on lui associe dans le graphe d'écart un arc de même sens avec la capacité restante. De plus, si un flot circule dans l'arc (u,v), on lui associe dans le graphe d'écart un arc de sens opposé avec la valeur de ce flot.

L'algorithme de Ford et Fulkerson

1. Initialisation : Considérer un flot compatible, par exemple le flot nul.
2. Construire le graphe d'écart correspondant.
3. Rechercher un chemin allant de la source vers le puits dans le graphe d'écart.
4. Deux possibilités sont à considérer :

- S'il existe un tel chemin, augmenter le flot circulant sur ce chemin dans le réseau du minimum des valuations des arcs du chemin dans le graphe d'écart. Recommencer alors à l'étape 2.
- Sinon, l'algorithme se termine et le flot courant est maximal.

L'algorithme de Ford et Fulkerson permet donc de déterminer un flot maximal dans un réseau de transport.

Liens	Fonction
https://www.youtube.com/watch?v=TgT5MYWI2n4	Cours sur les flots
https://www.youtube.com/watch?v=eL3fTl4mykY	Ford-Fulkerson pour construire un flot max.dans un graphe
https://www.youtube.com/watch?v=2A6OJ_tVt8o	Application : affecter des taches à des personnes (graphes bipartis)
https://www.youtube.com/watch?v=AJdRuNj9-r4	Flots 4 : algorithme de Ford-Fulkerson et choix des chaines améliorantes
https://www.youtube.com/watch?v=OF0f_fFbS7E	COURS FLOT MAXIMAL
http://cslndr.free.fr/GrapheTD6.htm	Exercices
« Introduction à l'algorithmique »,DUNOD, seconde édition	Livre
https://en.wikipedia.org/wiki/Ford–Fulkerson_algorithm	Algorithme et code (recopier le lien en entier)
https://www.youtube.com/watch?v=Tl90tNtKvxs	Exemple en anglais
https://www.youtube.com/watch?v=LdOnanfc5TM	Exemple en anglais
https://www.topcoder.com/community/competitive-programming/tutorials/maximum-flow-augmenting-path-algorithms-comparison/	Site internet
http://www.cs.yale.edu/homes/lans/readings/routing/ford-max_flow-1956.pdf	Article de LR Ford, DR Fulkerson, 1956

Objectif et cours : *Appliquer l'algorithme de Ford Fulkerkson et conceptualiser et modéliser un problèle de flot.*

Exercice 1

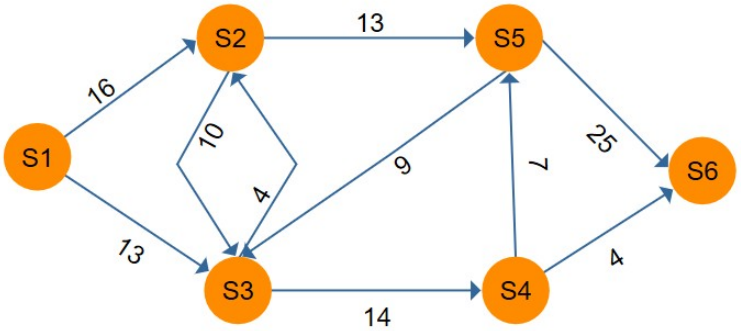
Question 1

Quelles est la complexité de l'algorithme Ford Fulkerkson ?

Question 2

Appliquez l'algorithme de Ford-Fulkerkson à l'exemples suivant :

1) *Sommet « source » S1, sommet « puits » S6*



Exercice 2

On cosidère un réseau de routage de telle sorte que Le serveur S est connecté à la machine M par un réseau avec les noeuds S1,S2,S3,S4, les capacités de connexions entre les noeuds sont (en Mbit/s) :

Système de routage					
	S1	**S2**	**S3**	**S4**	**M**
S	3	7	2		
S1		4		8	
S2				4	6

S3		3		7	
S4	4				5

On considère un utilisatuer x possédant une machine M téléchargeant un fichier volumineux du serveur S. L'objectif est de trouver le routage qui maximise le débit.

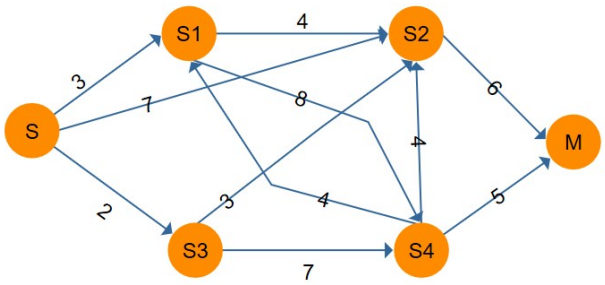
Question 1

A quel problème algorithmique correspond à ce problème de routage? Par quel algorithme pouvons-nous le résoudre ?

Question 2

Appliquez cet algorithme en détaillant chacune des étapes.

Aide pour démarrer : *Vous devriez obtenir le graphe suivant :*



Exercices supplémentaires

Question 1 (déjà corrigée en séance)

Ecrire un algorithme qui vérifie si un graphe non orienté possède un cycle.

Question 2

Soit G un graphe orienté qui possède n sommets numérotés de 1 à *n*n, et *m*m arcs numérotés de 1 à m. On appelle matrice d'incidence du graphe G, la matrice A=(ai,j) comportant n lignes et m colonnes telle que

- $a_{i,j}$ vaut +1, si l'arc numéroté j admet le sommet i comme origine;
- $a_{i,j}$ vaut -1, si l'arc numéroté j admet le sommet i comme arrivée;
- $a_{i,j}$ vaut 0 dans les autres cas.

a) Ecrire un algorithme de construction de la matrice d'incidence à partir de la liste d'adjacence d'un graphe, puis à partir de sa matrice d'adjacence.

b) Ecrire l'algorithme générique de parcours d'un graphe pour chacune des représentations. Quel est le temps d'execution pour chaque représentation?

Question 3

Si B^T désigne la transposée de B, que représente la matrice BB^T ?

Question 4

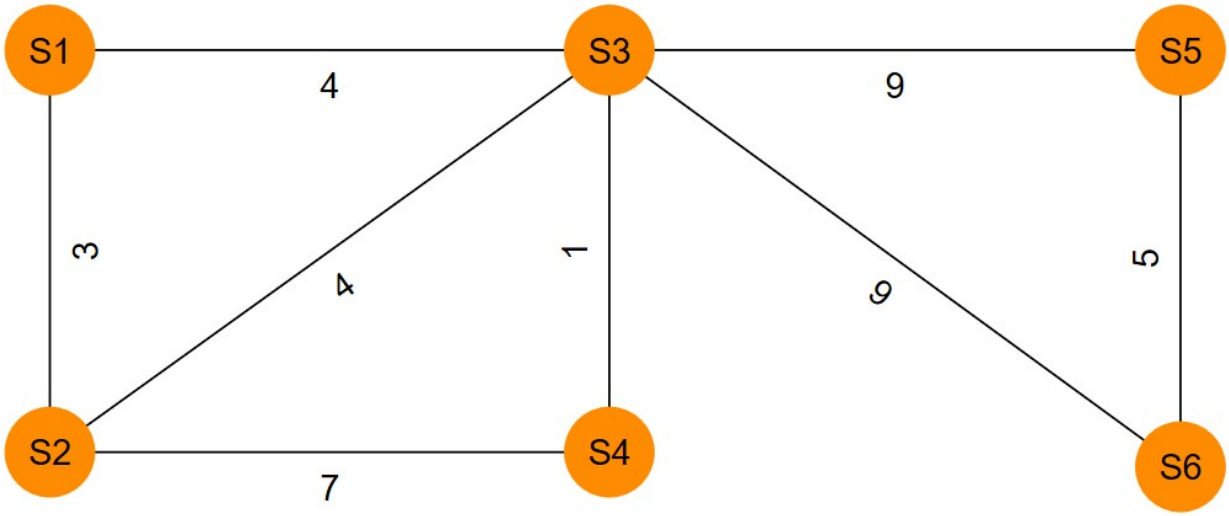
Considérons un graphe orienté G = (V,E). Un sommet v est un puits universel s'il est de degré entrant |V|−1 et de degré sortant 0. Etant donnée une représentation d'un graphe G = (V,E) par une matrice d'adjacence, proposer un algorithme permettant de déterminer s'il existe un puits universel.

Question 5

Modifier l'algorithme de parcours en profondeur donné en cours de telle sorte qu'il affiche pour chaque arc du graphe son type (arc de l'arbre de parcours, arc de retour, arc transversal, arc en avant).

Question 6 (déjà corrigée en séance)

Appliquer l'algorithme de Kruskal au graphe suivant :



Question 7 (déjà corrigée en séance)

Donnez un algorithme calculant l'ensemble des composantes connexes d'un graphe non orienté et calculez en la complexité.

Question 8

a) En reprenant l'algorithme de Kruskal du cours, on vous demande de décomposer cet algorithme en fonctions et/ou procédures.

b) Ecrire les algorithmes de toutes les fonctions et les procédures identifiées à la question a)

Question 9

a) *On vous demande de décomposer l'algorithme de Dijkstra en fonctions et/ou procédures.*
b) *Ecrire les algorithmes de toutes les fonctions et les procédures identifiées à la question a)*

Question 10 (déjà corrigée en séance)

Donnez un algorithme calculant l'ensemble des composantes connexes d'un graphe orienté et calculez en la complexité.

Corrections

Question 3 (correction)

La matrice M = BB^T = (mi,j) est carré de taille n.

- *Pour tout i ≠ j, $m_{i,j} = \sum_{k=1}^{nb\_aretes} b_{i,k} b_{j,k}$ vaut le nombre d'arêtes existant entre les sommets i et j (indépendamment de l'orientation)*
- *Pour tout i, $m_{i,i} = \sum_{k=1}^{nb\_aretes} b_{i,k} b_{i,k}$ vaut le degré de i.*

Question 8

a)

```
fonction Kruskal (gr : Graphe)
Variables locales
grAr : Graphe
listeArete : liste de Arete

créerGraphe(grAr)

enleverAretes(gr, grAr)                    // Procédure enleverAretes à implémenter :
                                           crée un graphe grAr avec les mêmes sommets
                                           que gr mais sans arêtes
listeArete <- trierAretes (gr)             // fonction trierAretes à implémenter :
                                           trie les arêtes par ordre croissant de
                                           valuation
créerComposanteConnexe(gr, composants)     // Procédure à implémenter : construit un
                                           tableau de composantes connexes de grAr
                                           indicés par les sommets
créerArbre(listeArete, composants, grAr)   // Procédure à implémenter : crée l'arbre
                                           couvrant de poids minimum correspondant au
                                           graphe gr
Retourner grAr
fin fonction
```

b)

```
procédure enleverAretes(gr: Graphe, grAr : Graphe (E/S))
Variables locales
ls : Liste de Element
s : Element

recSommets(gr : Graphe, ls)
Tant que non Vide(ls)
      s <- tete(ls)
      ajouterSommet(grAr, s)
      ls <- reste(ls
fin tant que
fin procedure

fonction trierAretes (gr: Graphe)
Variables locales
listeArete, listeArete1: liste de Arete
ar<- Arete

recAretes(gr, listeArete)
Tant que non Vide(listeArete)
      ar<-minimum(listeArete)
      ajouter(listeArete1, ar)
      listeArete<-reste(listeArete)
fin Tant que
retourner listeArete1
fin fonction
```

```
fonction minimum (listeArete : liste de Arete) : Arete
Variables locales
ar, areteMin : Arete
coutMin : Entier

coutMin <- recValuation(tete(listeArete))
Tant que non Vide(listeArete)
      ar<-tete(listeArete)
      Si recValuation(ar) < coutMin Alors
            areteMin<-ar
            coutMin<-recValuation(ar)
      fin Si
      listeArete<-reste(listeArete)
fin Tant que
retourner areteMin
fin fonction
```

```
procédure créerComposanteConnexe(gr: Graphe, composants : Map <Element> de Entier
(E/S))
Variables locales
sommets : liste de Element
i : Entier

creerMap(composants)
recSommets(gr, sommets)
i <- 1
// Première étape, numéroter les sommets
Tant que non estVide(sommets) Faire
    // On insère le couple (sommet courant, indice) dans la map composants
      stocker(composants, tete(sommets), i)
      sommets <- reste(sommets)
      i<-i+1
Fin tant que
fin procédure
```

```
procédure créerArbre(listeArete : liste de Arete, composants : Map <Element> de
Entier, grAr : Graphe (E/S))
Variables locales
ar :Arete
indiceOr, indiceDes : Entier
clefs : liste de Element
clef : Element

Tant que non estVide(listeArete)
      ar <- tete (listeArete)
      indiceOr <- valeurDe(composants, recOrigine(ar))
      indiceDes <- valeurDe(composants, recDestination(ar))
      Si indiceOr <> indiceDes Alors
            modifierValeur(composants, recDestination(ar), indiceOr)
            ajouterArete(grAr, ar)
            // On met à jour l'ensemble des indices qui étaient égaux à iniceDes
      listeClefs(composants, clefs)
            Tant que nonVide(clefs)
                  clef <- tete(clefs)
                  Si valeurDe(clef) = indiceDes Alors
                        modifierValeur(composants, clef, indiceOr)
                  Fin si
      Fin si
      listeArete <- reste(listeArete)
fin Tant que
fin procédure
```

Question 9

a)

```
procédure Dijkstra (gr : Graphe, origine : Element, longueurs : Map<Element> de
Entier (S), peres : Map<Element> de Element (S))
Variables locales
pivot : Element
succ, sommets : liste de Element
sPrime : Ensemble de Element        // L'ensemble des noeuds déjà "traités" (ayant
                                    été "pivot")
i : Entier
longueurs : Map<Element> de Entier // La Map qui donne les longueurs des plus courts
                                    chemins découverts jusque-là
peres : Map<Element> de Element     // La Map qui donne les pères des noeuds

creerMap(longueurs)
creerMap(peres)
creerEnsemble(sPrime)
pivot <- origine
sommets<-recSommets(gr)
initialiserLongueurs(origine, sommets, longueurs)
Pour i<-1 à taille(sommets)-1 faire
      recSuccesseurs(pivot, succ)
      miseAjourLongueurs(gr, succ, sPrime, longueurs, peres, pivot)
fin pour
fin procédure
```

b)

```
procédure initialiserLongueurs(origine : Element, sommets : liste de Element,
longueurs : Map<Element> de Entier (E/S))
Variables locales
s :Element
stocker(longueurs, origine, 0)
Tant que non Vide(sommets) faire
      s<-tete(sommets)
      si s<>origine Alors
            stocker(longueurs, s, infini)
      fin si
      sommets<-reste(sommets)
Fin tant que
Fin procédure
```

```
procédure miseAjourLongueurs(gr : Graphe, succ : liste de Element, sPrime : Ensemble
de Element, longueurs : Map<Element> de Entier (E/S), peres : Map<Element> de Element
(E/S), pivot : Element(E/S))
Variable locales
s : Element
ar : Arete
valeurAr, valeurS, valeurPivot : Entier

ajouter(sPrime, pivot)
Tant que non Vide(succ) faire
      s<-tete(succ)
      Si non appartient(sPrime, s) Alors
            existeArete(gr, pivot, s, ar) // trouve l'arete ar à partir de son
                                          origine (pivot) et la destination (s)
            valeurAr<- recValuation(ar)
            valeurS<- recValeur(longueurs,s)
            valeurPivot<- recValeur(longueurs, pivot)
            Si (valeurPivot+valeurAr<valeurS) Alors
                  modifierValeur(longueurs, s, valeurPivot+valeurAr)
                  Si (clefExiste(peres, s)) Alors
                        modifierValeur(peres, s, pivot)
                        Sinon
                        stocker(peres, s, pivot)
                  Fin Si
            fin Si
      fin Si
      succ -> reste(succ)
fin tant que
pivot<-chercherLongueurMinimum(longueurs, sPrime)
fin procédure
```

```
fonction chercherLongueurMinimum(longueurs : Map <Element> de Entier, sPrime :
Ensemble de Element) : Element
Variables locales
clefs : liste de Element
s, sommetMin : Element
longueur, longueurMin : Entier

recClefs(longueurs, sommets)
longueurMin<-infini

Tant que non estVide(sommets) faire
      s<-tete(sommets)
      longueur = recValeur(longueurs, s)
      Si (longueur < longueurMin) et (non appartient(sPrime, s) Alors
            longueurMin<-longueur
            sommetMin <- s
      fin Si
      fin tant que
retourner sommetMin
fin fonction
```

Algorithme de Dijkstra

Support	Liens
Exemple détaillé	***https://www.maths-cours.fr/methode/algorithme-de-dijkstra-etape-par-etape/***
Exemple	http://yallouz.arie.free.fr/terminale_cours/graphes/graphes.php?page=g3
Exemple	https://www.normalesup.org/~dconduche/informatique/PT/Cours/Dijkstra.pdf
vidéo	*https://www.youtube.com/watch?v=JPeCmKFrKio*

Algorithme en Pseudo-code

Soit G un graphe *connexe* dont les arêtes sont pondérées par des nombres *positifs*.

NOTATIONS :

S la liste des sommets du graphe ;
s_0 le sommet du graphe à partir duquel on veut déterminer les plus courts chemins aux autres sommets ;
$l(x, y)$ le poids de l'arête entre deux sommets x et y ;
$\delta_s(x)$ la longueur d'un chemin du sommets s_0 au sommet x ;
$V^+(x)$ la liste des successeurs du sommet x ;
$p(x)$ le prédécesseur du sommet x ;
X liste des sommets restant à traiter ;
E liste des sommets déjà traités.

INITIALISATION :

Pour Chaque $x \in S$ **Faire** $\delta_s(x) \leftarrow \infty$ *On attribue un poids ∞ à chacun des sommets x*
$\delta_s(s_0) \leftarrow 0$ *Le poids du sommet s_0 est nul*
$X \leftarrow S$ *La liste des sommets restant à traiter est initialisée à S*
$E \leftarrow \emptyset$ *La liste des sommets déjà traités vide*

TRAITEMENT :

Tant que $X \neq \emptyset$ **Faire** *Tant que la liste des sommets restant à traiter n'est pas vide*
  Sélectionner dans la liste X le sommet x avec $\delta_s(x)$ minimum
  Retirer le sommet x de la liste X
  Ajouter le sommet x à la liste E
  Pour Chaque $y \in V^+(x) \cap X$ **Faire** *On examine tous les successeurs y du sommet x qui ne sont pas traités*
    Si $\delta_s(y) > \delta_s(x) + l(x, y)$ **Alors**
      $\delta_s(y) \leftarrow \delta_s(x) + l(x, y)$ *La distance du sommet s_0 au sommet y est minimale*
      $p(y) \leftarrow x$ *Le sommet x est le prédécesseur du sommet y*
    Fin Si
  Fin pour
Fin Tant que

TD - Graphe : Connexité

Une structure de données d'ensembles disjoints gère une collection de données C={C1,…,C}.

Ce type de collection nommé *tableau associatif*, dictionnaire ou *map* permet de ranger des objets en fonction d'une clef dans *une table de symboles.* La clef doit généralement respecter un certain nombre d'invariants pour être valide (valeur de hachage ou résultats de la comparaison par exemple).

Chaque élément d'un ensemble est représenté par un objet. Soit *x* un objet, on considère les opérations suivantes :

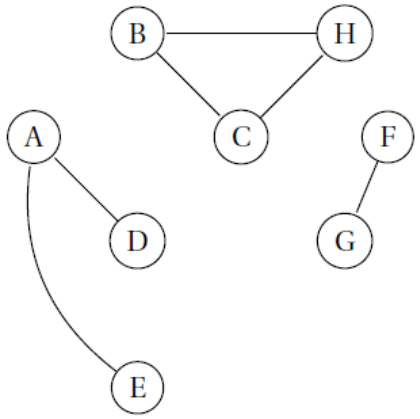
Création-MAP(x) : procédure permettant la création d'un ensemble dont le seul membre est l'objet x. S'agissant de la notion mathématique d'ensemble, l'objet x est unique et ne peut se retrouver dans un autre ensemble.

OPERARION-UNION(x,y) : procésure de réunion des senmbles dynamiques contenant les objets x et y. En notant les ensembles des objets contant x et y, Ex et Ey (respectivement), cette procédure créé un nouvel ensemble contenant l'union des deux ensembkles précédents disjoints. Les ensembles Ex et Ey sont par la suite détruits pour conserver la propriété d'unicité d'appartenance d'un objet à un ensemble.

TROUVER-MAP(x) retourne un pointeur vers le représentant de l'ensemble (unique) contenant l'objet x.

1. La notion de conteneur MAP est équivalent ici à la notion d'ensembnle (collection de couples (clé, valeur) : Les clés sont uniques : chaque clé est associée à une seule valeur).

<u>Détection des composantes connexes d'un graphe non orienté</u>



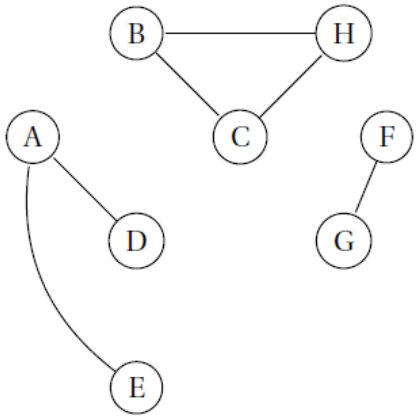
Sommet	Composante connexe
A	1
B	3
C	3
D	1
E	1
F	5
G	5
H	3

FIGURE 1 – Exemple de composantes connexes

Considérons la procédure **DETECTION-COMPOSANTES-CONNEXES(G)**, où G est le graphe considéré.

Fonction **DETECTION-COMPOSANTES-CONNEXES(G(V : sommets, A : Arêtes) : Graphe)** : Collection MAP

POUR chaque sommet u de G **FAIRE**
POUR chaque arête (u,v) de E **FAIRE**
SI TROUVER-MAP(u) ≠ TROUVER-MAP(v) **ALORS**
OPERARION-UNION(u,v)
//Création d'un nouveau MAP et destruction des deux anciens
FIN SI
FIN POUR
FIN POUR
FIN FONCTION



Sommet	Composante connexe
A	1
B	3
C	3
D	1
E	1
F	5
G	5
H	3

FIGURE 1 – Exemple de composantes connexes

Déroulement de l'algorithme

Arêtes traitées	**Collection d'ensembles disjoints**							
Ensembles initiaux	{A}	{B}	{C}	{D}	{E}	{F}	{G}	{H}
(A,D)	{A,D}	{B}	{C}		{E}	{F}	{G}	{H}
(B,C)	{A,D}	{B,C}			{E}	{F}	{G}	{H}
(F,G)	{A,D}	{B,C}			{E}	{F,G}		{H}
(A,E)	{A,D,E}	{B,C}				{F,G}		{H}
(B,H)	{A,D,E}	{B,C,H}				{F,G}		
(H,C)	{A,D,E}	{B,C,H}				{F,G}		

La procédure *DETECTION-COMPOSANTES-CONNEXES(G)* démarre avec chaque singleton et par union construit étape après étape les composantes connexes.

Support	**Liens**
Exemple détaillé	*https://www.youtube.com/watch?v=5wFyZJ8yH9Q*