TD algos & graph
Mathieu Liedloff
mathieu.liedloff@univ-orleans.fr
---12/09/12---

Exercice 1 : composantes connexes

Donnez un algorithme qui calcule, en temps linéaire, les composantes connexes d'un graphe non orienté.

Algos de parcours de graphes :
- Ford-Fulkerson
- Dijkstra
- parcours en largeur {
- parcours en profondeur{ O(n+m) : n = nb sommet, m = nb arrêtes.
- tri topologique (permet de savoir dans quel ordre réalisé des taches en respectant les contraintes de précédence)
- Prim {
- Krustal { trouver un arbre de poids minimum

Entrées :
une matrice d'adjacence du graphe

	A	B	C	D
A	0	1	0	0
B	1	0	1	1
C	0	1	0	1
D	0	1	1	0

Espace = $O(n^2)$ mais accès direct M[c,d]

ou liste d'adjacence :

A	B	/	
B	A	C	D/
C	B	D	/
D	B	C	/

Espace : O(n+2m), mais accès irréguliers

PARCOURS EN LARGEUR
les sommets ont 3 états :
- Blanc : non atteint
- Gris : atteint
- Noir = traité ;

Entrée : G(V,E)
début
- Pour chaque sommet v€V faire
  - v ← blanc ;
  - // on peut faire ici un compteur de composantes connexes
- Pour tout v€V faire // permet de parcourir plusieurs composantes

connexes
- si etat[v] = blanc alors
  - Ajouter v dans la file FIFO f
  - TANT QUE f!= null faire
    - x ← Extraire la tête de f ;
      - pour tout y € N(x) faire //N(x) : ensemble des voisins de x
        - si etat(y) = blanc alors
          - etat[y] ← gris ;
          - ajouter y à f ;
        - fin si
      - fin pour tout
    - etat[x] ← noir ;
  - fin TANTQUE
- finsi

- fin pourtout

fin

PARCOURS EN PROFONDEUR

debut

- pour chaque sommet v€V faire
  - etat[v] ← Blanc
- POURTOUT v€V faire
  - si etat[v] = blanc alors
    - ParcoursRec(v) ;
  - finsi
- fin pourtout

- ParcoursRec(x)
  - POURTOUT y€N(x) faire
    - si etat[y] = blanc alors
      - etat[y] ← gris ;
      - parcoursRec(y) ;
    - finsi
  - fin pourtout
  - etat[x] ← noir ;

fin

EXERCICE 2 : coupe minimum

Étant donné un flot maximum f dans un réseau R, proposez un algorithme qui calcule une de capacité minimal

Réseau : N = (V,A,s,p,c)
flot maximum f

(S,T) est une coupe ssi

- S U T = V et S inter T = null ; (autrement dit les ensembles S et T forment une partition de V)
- s€S

- p€P

//on observe que pour n'importe quelle coupe (S,T) on a nécessairement
|f| <= c(S,T)

//Théorème (Ford-Fulkerson en 1956)
Étant donné un réseau , la valeur du flot maximum est égal à la capacité d'une coupe minimum

//idée de démonstration
- prendre l'algo de Ford-Fulkerson
- s'il existe une coupe (S,T) tel que le flot F vérifie
  - |f| < c(S,T)
- alors on montre que le réseau résiduel admet un chemin améliorant(et donc que la valeur du flot f n'est pas maximum).

Pour tout (u,v) tel que u€S, v€T, f(u,v) = c(u,v)
car sinon il existerait un arc entre u et v dans le graphe résiduel
ainsi
(somme de tout (u,v) avec u€S et v€T de (f(u,v))= (somme de tout (u,v) avec u€S et v€T de(c(u,v))

Par conséquent , étant donné un flot maximum f dans un réseau, il suffit
- de calculer le réseau résiduel Gf //O(n+m)
- de calculer(par un parcours en largeur/profondeur) l'ensemble des sommets accessibles depuis s(on part de la source et on s'interdit d'emprunter un arc saturé) //O(n+m)

<u>Ford-Fulkerson</u>
Soit U= max{(u,v) : (u,v) soit un arc du réseau)}

complexité de l'algo : O(m*U)

<u>Edmunds- Karp</u>
ils calculent un plus court chemin , plutôt qu'un chemin améliorant, alors la complexité $O(n,m^2)$, pour cela il est conseillé d'utiliser un parcours en largeur.

<u>EXERCICE 3 :</u>
supposons que G = (V,E) soit un graphe non orienté connexe.
- proposez un algorithme qui détermine le plus petit nombre d' arrêtes a supprimer de G, de sorte à le rendre non connexe.
- Proposez un algorithme qui détermine le plus petit nombre de sommets(et donc d' arrêtes incidentes à ces sommets) à supprimer de G, de sorte à le rendre non connexe.

<u>Idée :</u> calculer une coupe minimum dans le réseau suivant....
on transforme le graphe en réseau

- quelque soit x € V
  - pour chaque y €V
    - construire le réseau où x est la source et y est le puits, pour tout

{u,v}€E, (u,v) et (v,u) sont des arcs de capacité 1
- calculer un flot maximum dans ce réseau
- obtenir la plus petite coupe obtenue

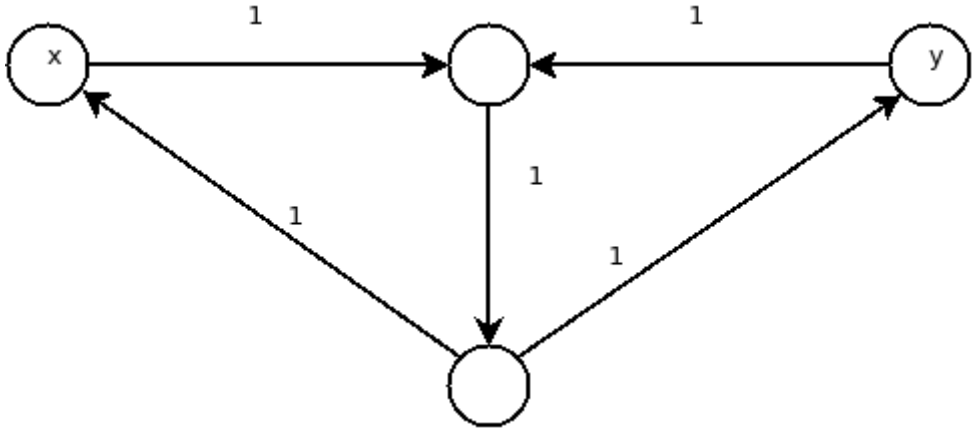
Exercice 3.2 : même chose mais pour supprimer des sommets

Idée : dans tous les cas , nous pouvons rendre non-connexe un graphe en utilisant l'algo de découpage par coupe minimale, on peut supposer que si on supprimer du graphe l'un des deux sommets des arcs a supprimer dans la coupe, il faudrait peut être choisir seulement un coté , soit seulement les sommets de gauche, soit ceux de droite. Toutefois , d'après l'algo de découpe minimale, on obtient les arcs a absolument supprimer, on choisit donc pour chaque arc a supprimer, le sommet qui à le moins d'arêtes incidentes, de manière a minimiser la somme des arêtes a supprimer.

19/09/12

Exercice 4 :
- Entrée : V : ensemble de sommets, E ensemble d'arcs, S : ensemble de sommets source, T : ensemble de sommets puits, |S| = |T| .
- Remarque , on peut se ramener à un problème de flots, en mettant la capacité de chaque arc à 1. Pour obtenir un réseau , on est forcé d'avoir une seule source et un seul puits, pour cela , on rajoute dans le réseau, un sommet s qui sera relié à toutes les sources, avec une capacité de 1 par arête, et un sommet t qui lui sera relié aux puits, de la même manière.
- S'il existe un flot maximum de p unités entre s et t, alors il existe p chemins arête-disjoints entre s1,...,sp et t1,...,tp.
- En effet , comme la capacité de chaque arc est de 1, alors il ne peux être utilisé que par un seul chemin, si la valeur du flot est p alors il existe p chemins
- Notons que si la valeur du flot est < à p alors le problème n'as pas de solution.

- 2. que se passe t il si le graphe précédent est non orienté ?
  - On construit un graphe en replaçant chaque arête par ce petit composant



- S'il l'on supprime les arcs d'une coupe minimum, on déconnecte le graphe
- par conséquent, si on supprime les arcs d'une coupe minimum du réseau précédemment construit alors on déconnecte ces sources des puits, et donc il n'existe plus de chemins entre ces sources et les puits.
- Si on ne supprime pas tous les arcs de la coupe minimum alors il existe au moins un chemin entre s et t, donc le nombre minimum d'arêtes à supprimer est donné par une coupe minimal.

- 4.

Exercice 5 : flot par échelonnement

soit G=(X,U) un réseau de sources et un puits p , les capacités c étant entières, On pose C = max{c(x,y)| (x,y) € U}.

- 1. montrer qu'une coupe minimum à une capacité au plus égale à C|U|
  - soit (S,T) une coupe quelconque du réseau, sa capacité est définie par
    - c(S,T) = ∑c(x,y) , (x,y)€U t.q. x€S, et y € T.
    - Comme la coupe traverse au plus m(où m = |U|) arcs et que la capacité de chaque arcs est bornée par C, il s'en suit que c(S ,T) <= m.C pour n'importe quelle coupe
- 2 . Pour un entier k donné, montrer que l'on peut calculer en O(n+m) un chemin améliorant de capacité au moins k si un tel chemin existe.
  - On construit le réseau résiduel , où la capacité résiduelle de chaque arc(u,v) est cf(u,v) = c(u,v) -f(u,v).
  - pour l'algo de Edmonds-Karp on a utilisé un parcours en largueur pour trouver un chemin améliorant
  - comme la capacité d'une chemin améliorant est égale à la plus petite capacité des arcs de ce chemin, s'il l'on souhait une chemin de capacité >= k, on n'a pas la droit d'emprunter un arc de capacité résiduel < k.
  - On va donc d'abord « nettoyer (O(n+m))» le réseau résiduel(en supprimant les arcs de capacité <k) avant de faire un parcours en largeur (O(n+m)) pour trouver un chemin améliorant(s'il existe).

- 4.
  - on observe que k est initialisé par une puissance de 2., ensuite dès qu'il n'existe plus un chemin améliorant de capacité au moins k, on divise k par 2, par conséquent, à la dernière itération de la boucle TantQue externe, la valeur de k est égale à 1. L'algorithme se termine lorsqu'il n'existe plus de chemin améliorant de capacité >= 1, autrement dit(comme dans Ford-Fulkerson) lorsqu'un flot maximum est trouvé.
- 5.
  - Montrons que la capacité résiduelle (cf(u,v) = c(u,v) -f(u,v).) d'une coupe minimum est au plus 2*k*m à chaque (début d') itération du TantQue externe.

- A la première itération, avant même la recherche du premier chemin améliorant, on a
  - (k = 2 ^ ⌊log C⌋) <= (c = 2^log C) <= (2*k = 2 ^ (⌊log C⌋+1)).
  - d'apres la question 1 , la capacité d'une coupe minimum est qu plus C.m donc C.m <= 2km
- Regardons la i-ème itération
  - à la fin de la i-1_èmem itération il n'existait plus de chemin améliorant de capacité >= k' , avec k' = 2k.
  - Soit S le sous ensemble de sommets t.q. Pour tout v €S, il existe un chemin de sà v de capacité >= k' dans le réseau résiduel.
  - Comme la (i-1)ème itération et terminée , il n'existe plus de chemin entre s et t.
  - Donc (S, V\S) est une coupe .
  - De plus tous les arcs(il y en a au plus m) de cette coupe ont une capacité <= k'
  - Donc la capacité de cette coupe est < m*k'
  - Ainsi la capacité résiduelle de la coupe minimum au début de la ième itération est au plus mk' = 2mk.
- 6,montrer que la boucle tant que interne est exécutée au plus O(m) fois pour chaque valeur de k :
-
  - 1 :on a montré que la capacité résiduelle d'une coupe est <= 2km
  - 2 : il n'existe pas de chemin améliorant de capacité >2k(sinon il aurai été trouvé à l'itération précédente du tant que externe
  - 3 à cause de la condition de la boucle tantQue interne, nous cherchons seulement des chemins améliorant de capacité supérieure ou égale à k
  - donc d'après 2&3 les chemins trouvés dans cette itération de la boucle externe ont des capacités comprises entre k et 2k
  - donc par 1 on en déduit que le flot ne peux être augmenté de plus de 2km
  - donc pour chaque valeur de k, la boucle va tourner m fois,donc en tout on a O(m).
- 7 complexité de l'algorithme
  - recherche c max => O(m)
  - init : O(m)
  - boucle externe : log(C)
  - boucle interne : O(m)
    - dedans :
      - recherche chemin:O(n+m)
      - augmenter flot O(m)
  - total : O(n+m+logc* m*(m+n))
    - O(m(n+m).log(C))
    - O(m².log(C))

26/09/12

Planarité :

Un graphe st dit planaire, si il peut être plongé (dessiné) dans le plan ,

sans qu'aucune arrête ne se croisent.

Théorème de Kuratowski(1896-1980)

Un graphe est dit planaire si et seulement si il ne contient ni le K5 ni le K3-3, comme mineur.

Mineur :

Soient G et H deux graphes.

On dit que H est mineur de G si H peut être obtenu en appliquant sur G les opérations suivantes :

- supprimer un sommet et ces arrêtes incidentes
- suppression d'une arête.
- contraction d'une arête

TD2 :

Exercice 1 :

- 2 :
- Soit T un arbre couvrant de G.
- notons n le nombre de sommets de G, et par mg son nombre d'arêtes
- (a)Notons mt le nombre d'arêtes de l'arbre T
  - on a n= mt+1(dans un arbre le nombre de sommets est toujours égal au nombre d'arêtes +1
- Soit G* le graphe dual de G : à chaque face de g , on associe un sommet. Deux sommets sont voisins si leur face ont au moins un arête en commun. Dans le cas ou plusieurs arêtes sont communes, on crée des arêtes multiples.
- (b)Soit T* inclus dans E* les arêtes du graphe dual G*, correspondant aux arêtes de E\T.
  - On observe que les arêtes de T* relient toutes les faces, car T n'as pas de cycle(c'est un arbre).
  - De plus T* n'as pas de cycle, sinon il séparerait des sommets de G à l'intérieur de ce cycle , et donc T ne serait pas un arbre couvrant.
  - Donc T*est un arbre couvrant de G*
- © f = mt* +1
  - le nombre de sommets dans G* (et donc dans T*) correspond au nombre de faces de G(par construction) +1
- (d) n = mt+1
  - f + n = mt* +1 + mt +1
  - or mt + mt* = m
  - donc f+n = m+2
  - donc n-m+f = 2 (formule d'Euler).

Test avec un graphe K5

montros que le K5 n'est pas planaire.

Supposons que K5 soit planaire. Donc il vérifie la formule d'Euler n-m+f= 2

- dans le K5
  - n= 5 , m = n*(n-1)/2 = 10
  - la formule d'Euler nous apprend que f =2-n+m = 2-5+10 = 7

- une face est composée d'au moins trois arêtes, et chaque arête sert pour deux faces.
- Docn le nombre d'arêtes est au moins 3*f/2 = 3*7+2 = 10,5
- or 10,5 > 10(nb arêtes dans K5).
- Ainsi le K5 est connexe mais ne respecte pas la formule d'Euler, donc le K5 n'est pas pas planaire

Genre des surface :

Genre 0 : sphère ou plan
Genre 1 : tore : on a jouter une anse à la sphère pour obtenir le tore

le genre d'une surface connexe est le nombre maximum de courbes fermées simples sans points communs que l'on peut tracer à l'intérieur de cette surface sans la déconnecter.

3/10/12

essayons de montrer que le K5 peut être plongé dans un tore. -done.

n-m+f = 2-2*g, le genre g d'un graphe est le plus petit genre d'une surface su rl'aquelle le graphe peut être plongé

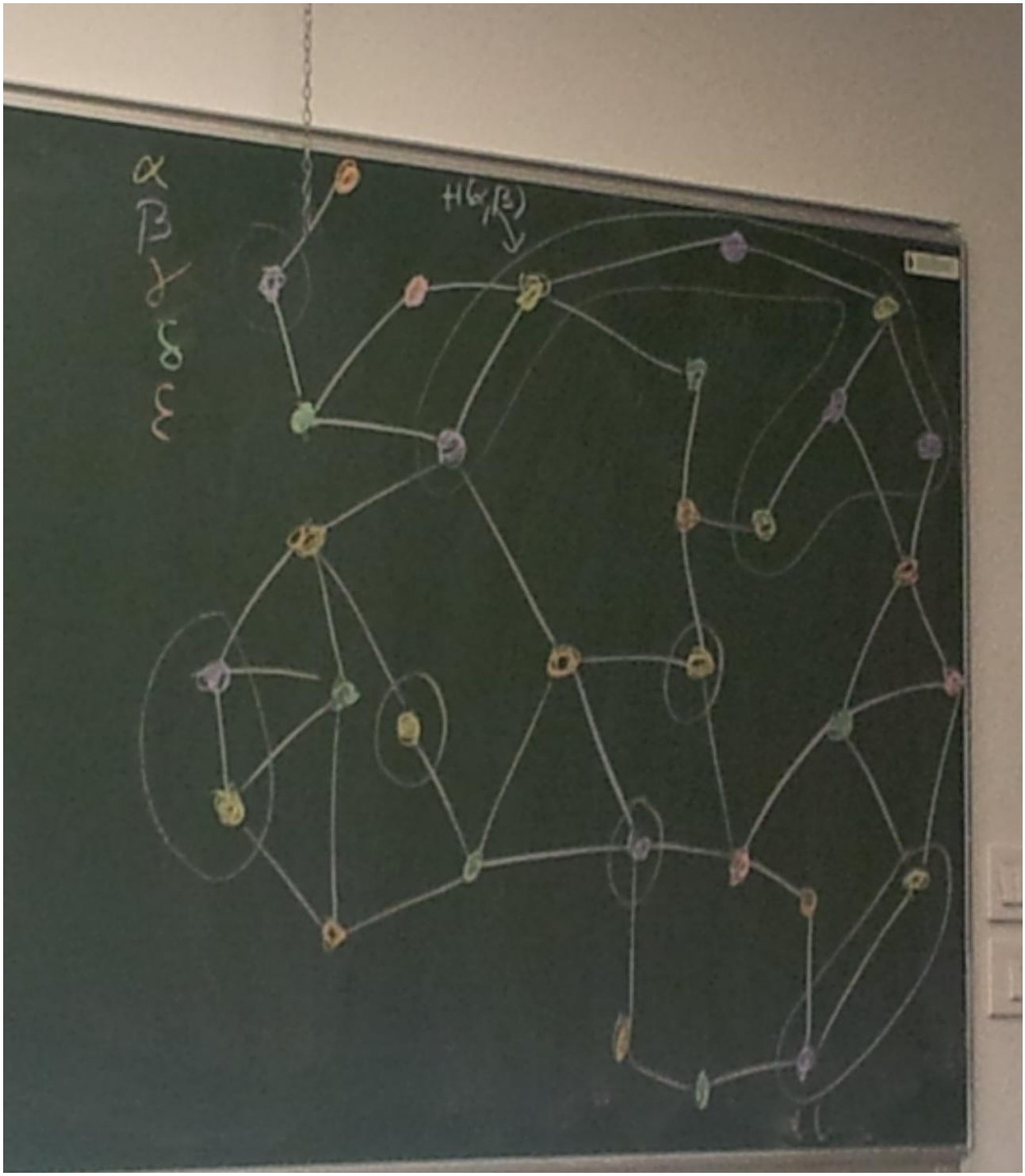
à l'ai de de la seconde formule d'Euler, donner le genre du graphe complet à n sommet(noté Kn), en fonction de n

- soit n-m+f = 2-2*g
- soit n – (n*(n-1)/2) + f = 2- 2*g
- g = [2-n+(n*(n-1)/2) -f]/2
- chaque arête sert dans deux faces,
- chaque sommets apparaît dans n-1 faces, et chaque faces contient exactement trois sommets.
- Puisque G est un graphe complet, le graphe est complètement triangulé, c’est à dire qu'il n'existe pas de cycle(induit) de longueur >=4 . Autrement dit , un cycle de longueur >= 4 contient au moins une corde(raccourci).
- g = [2-n+(n*(n-1)/2) -f]/2, f= n*(n-1)/3
- g = [2-n+(n*(n-1)/2) – (n(n-1)/3)]/2
- g(Kn) = (n-3)(n-4)/12 // n >= 3

Exercice 2 : 5-coloration des graphes planaires

- 1,
  - Soit g un graphe quelconque. Considérons une coloration de G et deux couleurs a et b utilisées dans la coloration .
  - Soit G un graphe coloré
  - soit G(a,b) le sous graphe induit par les sommets colorés a et b. Notons H(a,b) une composante connexe de G .
  - on observe qu'il est possible d'échanger les couleurs a et b dans H(a,b). en effet d'une part, pour tout sommet n'appartenant pas à H mais ayant un voisin dans H, sa couleur n'est ni a, ni b.
  - D'autre part dans H(a,b) où on permute les deux couleurs a et b, on n'a

jamais 2 sommets voisins avec la même couleur car le graphe était initialement bien coloré.



- 2,
  - Soit G un graphe planaire et x un sommet ayant 5 voisins. On orient e e voisinage de x dans le sens trigonométrique, les voisins sont A,B,C,D,E
  - G' =G -x, construisons une coloration de G', supposons que A,B,C,D et E sont de couleurs différentes, a,b,c,d,e.
    - Montrons que l'on ne peux pas avoir
      - I) A et C dans une même composante connexe de G'(a,c).
      - II) ET B et D dans un même composante connexe de G'(b,d).

    - Supposons que ce soit le cas. Donc il existe un chemin de A à C qui n'utilise que des sommets colorés avec a et c.
    - Par ailleurs, on a aussi un chemin entre B et D colorés b et d.
    - Comme le sommets b est à l'intérieur du cycle XA....C et que le sommet d est extérieur à ce cycle. Il n'est pas possible d'avoir un chemin bicolorié b et d de B à D.(car le graphe est planaire).
- Montrons qu'il existe une coloration de G' ou A,B,C,D et d utilisent seulement 4 couleurs
    - comme montré à la question précédentes, au moins l'une des conditions I ou II n'est pas remplie
    - Sans perte de généralité, supposons que la condition II n'est pas remplie , donc B et D sont dans deux composantes connexe différentes, du graphe induit par les couleurs b et d.
    - Donc par la question 1 , si on permute les couleurs de la composante connexe contenant B, B sera colorié en d et donc A,B,C,D et E n'utiliserons plus que 4 couleurs différentes.
- On viens de montrer que pour tout sommet de degré 5 , ils est possible de colorer ses voisins avec seulement 4 couleurs et donc de colorer ce sommet avec la 5ième couleur.
    - Remarque : pour les sommets de degré <=4 c'est encore plus facile puisque leurs voisins utilisent au plus 4 couleur et il reste donc au moins une couleur disponible pour x).
    - Ainsi , pour tout graphe planaire de degré <=5 , ce graphe est 5-colorable.
- Montrons par induction(sur le nombre de sommets) qui tout graphe planaire est 5-colorable
    - Cas de base :
        - si n <= 6 alors le degré du graphe est <= 5 . donc par les questions précédentes il est bien 5-colorable.
    - Induction :
        - hypothèse, tout graphe planaire de degré n-1 est 5-colorable.
        - Soit G un graphe planaire à n sommet, montrons que g est 5-colorable
            - soit d(G) = somme(d(x))/n : le degré moyen du graphe. = 2m / n
        - Lemme : dans un graphe planaire m<= 3n-6
        - Ainsi d(G) <= 2*(3n-6)/n = 6n/n-12/n <= 6
        - Donc g contient au moins un sommet de degré au plus 5.
        - Par hypothèse d'induction , G-x contient n-1 sommet, et donc est 5-colorable
        - comme le degré de x est <= 5 , par les questions précédentes, il est possible de trouver une coloration de ses voisins avec au plus 4 couleurs, et donc de colorier x, donc G est 5-colorable .

10/10/12

théorème : Un graphe est eulèrien (c'est à dire qu'il admet un cycle eulérien) si et seulement si tous les sommets ont un degré pair.

- 2 soit G = (V,E) un graphe , sans sommet isolé, dont tous les sommets ont un degré pair, montrer que tout cheminement maximal est un cycle
- un cheminement maximal u = (x, … , y) est tel que toutes les arêtes incidentes à y soient dans u , c'est à dire que l'on ne peux plus prolonger le chemin depuis y.
- hypothèse ; pour tout x €V, d(x) est pair.
- On appelle x et y extrémités et le reste du chemin , la partie interne
- dans u quel est le nombre d'arêtes incidentes à y ?
  - dans la partie interne de u , il y en a un nombre pair, à chaque fois que l'on arrive sur y, on en repart.
  - Il y a aussi une arête incidente à y à la fin du cheminement u.
  - donc au total, il y a un nombre impair d'arêtes incidentes à y dans u .
- Par conséquent, il existe z tel que {y,z}!€ u.
  - si z!= x alors
    - u = x →z1→...→y→z est plus long que u, contradictoire avec la maximalité de u.
  - si z = x, alors u est un cycle. !!
- 3 soit s un sommet de N
  - soit Ci une composante connexe de G\V et soit t un sommet de Ci.
  - Comme le graphe est connexe, il existe un chemin de t à s.
  - Soit le premier sommet de ce chemin qui appartient à N . Ce sommet appartient à la fois à Ci et a N.
- 4 G =(V,E) un graphe connexe, montrer que
  - G est eulérien si et seulement si pour tout x€V, d(x) est pair.
    - Soit u un cycle eulérien de G, soit y un sommet de u
    - on note que dans u , à chaque fois que l'on arrive sur y , on en repart, comme u contient toutes les arêtes du graphe , alors d(y) est pair.
    - Supposons que tous les degré soient pairs.
      - Preuve par induction sur le nombre d'arêtes .
        - Cas de base : m = 0.
          - comme le graphe est connexe , alors g contient au plus un sommet , et il est donc trivialement eulérien.
        - Induction : m > 0
          - hypothèse d'induction, tout graphe qui a <m arêtes
            - Étant connexe et dont tous les sommets sont de degré pair, il est eulérien
          - Soit G un graphe ayant m arêtes, connexe, pour tout x €V, d(x) est pair, d'après la fabuleuse question 2, ce graphe à un cheminement v qui est un cycle
          - d'après l'extraordinaire question 3, il existe des composantes connexe de G-v qui admettent des sommets de v,
          - On note que pour tout i ,
          - Ci est connexe,
          - pour tout x € Ci , d(x) est pair car même si l'on enlève deux arêtes incidentes d(x) reste pair.
          - De plus , Ci contient moins d'arêtes que G, par hypothèse d'induction, la composante Ci est eulérienne. Notons ui un

cycle eulérien de Ci.
- On observe que l'on peut « recoller » le cycles ui au cycle v.
- On obtient ainsi un cycle eulérien de G.

- 5 déduire un algorithme qui teste si un graphe est eulérien et qui construit le cycle le cas échéant
  - Eulérien(G)
    - vérifier que G est connexe et que ¥x , d(x) est pair.
    - Construire cheminement maximal v dans G(Q1)
    - si V ne contient pas toutes les arêtes de G faire
      - soit C1, ..., Cp des composantes connexes de G\v(Q2)
      - ¥ composante connexe Ci faire
        - µi ← eulérien (G[Ci])
      - fusionner v et µi(Q3)
    - finsi
    - retourner v .

Un graphe est dit semi-eulérien , si il existe une chaîne non -fermée passant une fois et uen seule par chaque arête.

17/10/12

Exercice 3 (in)approximation

Comment résoudre Cycle Hamiltonien en utilisant TSP-approx ?

- TSP-approx attend comme paramètres :
  - un graphe complet , GB.
  - une distance d(u,v) entre chaque sommets u et v.
- une instance de graphe hamiltonien
  - un graphe GA (pas nécessairement complet).
  - On doit transformer Ga en Gb,db tel que Ga admet un cycle hamiltonien
    - on doit compléter le graphe
    - pour les arêtes originales , on met le poids à 1
    - pour les nouvelles arêtes , on met un poids de l'infini
    - on exécute TSP-approx sur le nouveau Ga
    - si TSP-approx nous rend une solution non infinie, alors on a un cycle hamiltonien de poids n, donc TSP-approx ne peut qu'utiliser un cycle hamiltonien (sinon il retourne une solution infinie qui est bien plus grande qu'une 10-approximation)
    - Inversement si TSP6approx trouve une solution infinie, c'est que Ga n'a pas de cycle hamiltonien, sinon si cette solution est de poids n , alors le cycle est hamiltonien
-

Exercice 5 : Algorithmes Galactiques
on a G= (V,E) où V= {v1,v2,v3,...,vn}
Sans perte de généralité, supposons que le cycle (qui est la solution du TSP) commence au sommet v1.

Pour tout S << {v2,..,vn} et vi € S.
On pose Opt[S,vi] comme étant le poids d'un chemin de plus petit poids qui commence a v1, passe exactement par tous les sommets de S et qui se termine à vi

On va calculer Opt[S,vi], pour tout S et pour tout vi en considérant les ensembles S par ordre de cardinalité croissante.

Opt[S,vi] = min(vj €S\{vi}) Opt[S\{vi},vj] + d(vj,vi).
La solution optimale au TSP sera donnée par min(vk € {v2,...,vn})Opt[{v2,v3,...,vn},vk] + d(vk,v1).

Temps d'exécution
O(2^(n-1).(n-1).(n-2)).

Exercice 6 : Chemin Hamiltonien dans les tournois
1- montrons par induction sur le nombre de sommet, qu'un tournoi possède toujours un chemin hamiltonien

- Soit G un tournoi
- cas de base : n=1
  - trivialement le graphe admet un chemin hamiltonien
- Induction : n>2
  - hypothèse d'induction : tous graphe qui a n' <n sommet admet un chemin hamiltonien
  - montrez que la propriété est encore vraie au rang suivant
    - Soit G un tournoi à n sommet
    - montrons que G admet un chemin hamiltonien
    - Soit u un sommet quelconque de G.
    - comme G\{u} est un graphe complet orienté ayant n-1 sommet, par hypothèse, il admet un chemin hamiltonien
    - Notons ce chemin par :
      - v1→v2→v3→v4→...→vn-1
    - Soit i(1≤i≤n-1) le plus grand indice tel que pour tout j≤i , (vj,u) est un arc de G.
    - Mézalor, (vi+1,)!€G et donc (u,vi-1)€G,
    - on peut donc remplacer l'arc vi→vi+1 dans le chemin , par vi→u→vi+1
    - et donc G ayant n sommet admet un chemin hamiltonien

de cette preuve , on peut construire un algorithme incrémental
- G[v1] → chemin hamiltonien v1
- G[v1,v2]→ trouver où insérer v2→ chemin hamiltonien
- ...
- G(V) → ou insérer vn→chemin

24/10/12

Algorithmes probabilistes

fiche 4

EX1 : collection de vignettes

collection de n vignettes : combien de paquets pour avoir toutes les vignettes

- bien sur ce nombre est ≥ 20.
- Au début , la probabilité d'avoir une nouvelle vignette est grande, mais si on a déjà beaucoup de vignettes, la probabilité d'avoir une nouvelle vignette s'amoindrit
- par exemple si j'ai trouvé n-1 vignettes, alors la probabilité de trouver la dernière vignette est de 1/n.
- Soit X la variable aléatoire définie par :
  - X = nombre de paquets à acheter pour obtenir toutes les n vignettes.
- Pour faciliter le calcul de E(X), qui est la valeur attendue que l'on cherche, on va remplacer X par la somme des variables Xi
- Si on a déjà trouvé j vignettes, la probabilité de trouver une nouvelle vignette dans le prochain paquet est : (n-j)/n
- soit Xj le nombre de paquets achetés entre le moment où l'on a j vignettes et le moment ou l'on en trouve une nouvelle.
- Donc X= X0+X1+...+Xn-1 // nombre de paquets achetés pour avoir la n-ième vignette.
- On doit calculer l'espérance E(X).
- E(X) = E(Σ Xj) = Σ E(Xj).
  - E(Xj) = Σ(k = 0 → + inf) k × P(Xj = k) = Σ k × (1-p)^(k-1)/* probabilité d'acheter un paquets avce une vignette que l'on a déjà, événement complémentaire*/ × p
  - théorème Σ k × (1-p)^k = (1-p)/p²
  - preuve : Σ(k : 1->inf) k .a^k = a +a²+a³ + ... + a^k
  - ainsi (1-a)/a × Σ k.a^k
    - = (1-a)+ (2a-2a²)+ ... +
    - = 1 +a +a² + a³ + ...
    - Or Σ(k:0->inf) a^k = 1/(1-a)
    - D'où Σ(k:1->inf) k . a^k = 1/(1-a.(a/1-a) = a/(1-a)²
- Ainsi E(Xj)
  - = Σ(k :0->inf) k .(1-p)^(k-1) p //on veux une puissance k et on commence à un car on connais le cas ou k =0
  - = p/1-p * Σ(k:1->inf) k*(1-p)^k
  - = p/1-p * 1-p/p² = 1/p = n/n-j
- Donc au total
  - E(X) = Σ E(Xj) = Σ n/n-j = n*Σ 1/n-j = n . Σ 1/i = n * H(n)
  - or H = O(ln(n)
  - d'où E(X) = n.H(n) = O(n.ln(N))

Exercice 2

- 1 – temps d'exécution au pire cas :
  - Au pire des cas, l'élément choisi à la ligne 4 est le maximum de A. Dans ce cas, la ligne 9 s'exécute et x est comparée avec les éléments de A, le temps d'exécution est décrit par
    - $\Sigma i = n(n+1)/2 = O(n^2)$.
- 2 – temps d'exécution attendu de l'algorithme
  - soit Xk la variable aléatoire, qui étant donné un ensemble A de cardinalité k prend la valeur n si l'élément choisit à la ligne 4 est le maximum et qui prend la valeur 1 sinon
  - Xk représente le temps d'exécution d'un appel récursif
  - si X représente le temps total , $X = \Sigma Xk$
  - $E(Xk) = \Sigma P(Xk = j) = 1*P(xk = 1) + k * P(Xk = k)$
    - $1.(1-1/k)+k*1/k$
    - $k-1/k+k/k$
    - $< k/k+1 = O(1)$
  - $E(X) = \Sigma 1->n (E(Xk) \leq v \Sigma O(1) = O(n)$.
  - donc le temps attendu est O(n).

Exercice 3 : 3-coloration

- 1 – proposez un algorithme polynomial probabiliste qui calcule une 3-coloration
  - Entrée :
    - un graphe G(V,E), ayant n sommets et m arêtes
    - algo MonJoliAlgorithmeAvecDesCouleurs
      - pour chaque sommet v du graphe faire(
        - f(v) ← Random(Rouge,Vert,Bleu).
  - Soit X la variable aléatoire égale au nombre d'arêtes valides dans un une coloration produite par l'algorithme.
  - Pour chaque arête (u,v) de G on définit la V.A Xuv
    - = 1 si (u,v) est valide)
    - = 0 sinon
  - ainsi $X = \Sigma xuv$
  - le nombre attendu d'arêtes valide est donnée par l'espérance de X
  - $E(X) = E(\Sigma xuv) = \Sigma E(Xuv) = \Sigma k P(Xuv=x) = m.2/3$
  - $E(Xuv)= 0 * P(Xuv=0) + 1* P(Xuv=1) = P(Xuv= 1). = 2/3$

<u>Exercice 4 :</u>

- 1/ on considère un algorithme qui choisit aléatoirement si il met le sommet v dans A ou dans B
  - a/ on cherche a savoir le nombre d'arêtes attendu
    - soit X la VA qui représente le nombre d'arêtes attendu :
    - $X = \Sigma Xuv$
    - Xuv = 1 si u est dans un des deux ensembles et v dans l'autre , et 0 sinon
    - $E(X) = E(\Sigma Xuv) = \Sigma E(Xuv) = \Sigma k (P(Xuv) = k) = 1\times P(Xuv = 1) = P(u \in A)\times P(v\in B) + P(u\in B)\times P(v\in A) = 1/2\times 1/2 + 1/2\times 1/2 = 1/2$ et puisque les élément sont indépendants , on peut multiplier par

le nombre d'arêtes total pour avoir le nombre d'arêtes attendues soit m×1/2.

- b/
  - la question précédente garantie que pour n'importe quel graphe , il existe au moins une coupe ayant au moins m/2 arêtes.

- 2/
  - c/ Montrez que le rapport d'approximation de cet algorithme est au plus de 2.
    - notons par SOL le nombre d'arêtes traversant la coupe dans une solution retournée par l'algo, et notons opt , le nombre d'arête d'une coupe optimale.
    - On veux montrer que 1/2×OPT ≤ SOL ≤OPT
    - Étant donné un sommet vi , notons son voisinage N(vi)
      - N(vi) = {vj tels que {vi, vj}€E}
      - le degré de vi est noté d(vi)=|N(vi)|
    - Voisinage partiel :
      - ~N = voisins de vi dont l'indice est inférieur à i
      - ~d(vi) = |~N|
    - on sait que
      - Σ d(vi) = 2m
      - Σ ~d(vi) = m
      - ligne 4 de l'algorithme
        - (*) Si | ~N(vi) inter A | ≥ | ~N(vi) inter B | alors vi est ajouté a b, sinon vi est ajouté a A
      - notons dans la suite
        - mAB le nombre d'arêtes quit raversent (A,B)
        - mA le nombre d'arêtes totalement dans A.
        - mB ---------- B.
        - miAB le nombre d'arêtes qui traversent AB ayant pour extrémités vi.
        - miA le nombre d'arêtes totalement dans A ayant pour extrémité vi.
        - mib le nombre d'arêtes totalement dans B qui ont pour extrémité vi
      - À cause du choix (*) fais par l'algorithme
        - pour tout i ,
          - si vi €A alors miAB ≥miA et miB = 0(car vi €A)
          - ou si vi €B alors miAB ≥miB et miA = 0 (car vi €B)
        - donc pour tout i , miAB ≥ miA+miB
        - d'ou ΣmiAB ≥ ΣmiA+miB
        - donc mAB ≥ mA +mB
        - de plus si on note m le nombre d'arêtes du graphes
          - m = mAB + ma+ mb
          - donc m ≤ mAB + mAB
          - m ≤ 2mAB
          - m/2 ≤mAB
          - comme OPT ≤ m
          - OPT /2 ≤ mAB ← solution de notre algorithme.

Feuille 5 de TD

Exercice 1 : Chargement de conteneur

- 1 donnez ensemble de poids tels que l'algorithme n'utilise pas le plus petit nombre possible de camions
  - prenons la limite K à 100 , et un total de charge de 200 tel que {75,75,25,25} ,, l'algorithme glouton , va donc placer le premier 75 dans le camion 1, ensuite il tombe su r75 , c'est trop , le camion 1 est donc « chargé » et part, ensuite il place le 75 dans un camion n°2, il peux encore placer le 25 en position 3, et place finalement le dernier 25 dans un camion n°3, alors qu'il aurait été possible de placer le dernier 25 dans le camion n°1
- 2 montrez que le nombre de camion est au pire deux fois plus un facteur 2 du plus petit nombre possible
  - il apparaît deux cas :
    - soit le chargement w est placer dans un le bon camion,
    - soit il est placer dans un nouveau camion alors qu'il pourrait être placer dans un autre camion
    - soit A le nombre de camion renvoyé par notre algorithme
    - soit Opt le plus petit nombre de camion nécessaire
    - Montrons que A/Opt ≤ 2
    - notons que Opt ≥ Σwi / K (*) (dans le cas de l'égalité , tous les camions sont chargés au maximum)
    - Lemme : Pour chaque paire consécutive de camions retournés par notre algorithme, la somme des poids des conteneurs des deux camions est > K. Preuve : Trivial !
    - Par contradiction supposons qu'il y ait deux camions consécutifs dont la somme des poids transportés soit ≤ K
    - L'algorithme aurait placé des conteneurs les conteneurs du second camino dans le premier puisque cela ne dépasse pas K
    - Supposons que A soit pair
      - comme il y a A/2 paires de camions qui transportent un poids strictement supérieur à K dans la solution retournée par l'algorithme ; et en plus chaque conteneur doit être transporté par l'un des camions de la solution , on en déduit que Σwi > A/2 * K , par al propriété (*) on a K.opt ≥ Σwi , donc opt > A/2
    - Supposons que A soit impair
      - Σwi > A/2 *K > (A-1)/2 *k
      - (*) K *opt ≥ Σwi
      - donc K*Opt > (A-1)/2 * k <=> 2*opt > A-1 <=> 2*opt ≥ A comme A et Opt sont des entiers.
- Exercice 2
  - Montrons que l'algorithme « Répartir-Taches » est un algorithme d'approximation de rapport 2 .
  - Notons Opt la charge de travail maximum (sur toutes les machines)

dans une solution optimum.
- 2 observations
  - (*) Opt ≥ Σ ti
  - (**) Opt ≥ ti (1 ≤i ≤ n)
- Notons Mj la machine ayant la plus grande charge de travail dans un solution retournée par l'algorithme . Notons Tj sa durée totale de travail , et notons ð la dernière tache qui lui a été affecté, et tð sa durée
- À cause du choix glouton de l'algorithme ,
  - pour tout i entre 1 et 10 i ≠ j on a
    - Ti ≥ Tj – Tð (toutes les autres machines sont plus longues que Mj moins sa dernière taches) (car dans l'algorithme Tj- Tð est la machine avec la plus petite charge)
    - si on fais « la somme » de toutes les taches , on obtient
      - Σti ≥ 10.(Tj – Tð) <=> TJ – tð ≤ Σti/10 ≤ opt (par) <=> Tj ≤ Opt + tð <=> Tj ≤ 2 Opt(par **) .
- Lemme : à chaque étape d'itération de l'algorithme, la différence entre Ti = min 1≤k≤10 Tk
- et Tj = max Tk est au plus égale à 50.
- car on apprend dans la démonstration précédente que Ti-Tj ≤ Tð ≤ 50
- Donc en particulier à la fin de l'exécution de l'algorithme, on a
  - Ti-Tj ≤ 50 <=> tj ≤ 50+Ti <=> <*.*< ^*.*^ >*.*> v*.*v
  - Tj ≤ 50 +Σti /10 <=> Tj ≤ 50+3000/10 = 350 or 300+20 % = 360

28/11/12
fiche 6
Exercice 1 : tout ce qui est stable est contestable
entrée : un graphe G=(V,E) et un entier k ;
sortie : existe t il un sous-ensemble de sommet S⊂V tels que |S| ≥ k et pour tout u et v € S , u et v ne soient pas adjacents ? Un tel ensemble est dit stable.

- formaliser un algorithme pour invite le plus d'amis sans avoir deux amis qui ne s'entendent pas .
  - Pour cela, nous allons créer un graphe avec tous les amis de la liste , en mettant des arêtes entre les amis qui ne s'entendent pas. Et ensuite, on résout le problème d'ensemble stable
- Modélisons le problème du stable maximum par un programme linéaire(entier)
  - pour cela nous allons créer une variable par ami
  - on associe à chaque sommet vi du graphe une variable xi de valeur
    - xi = 0 → vi n'appartient pas au stable
    - xi = 1 → vi est dans le stable
  - OBJECTIF : MAXIMISER x1+x2+…xn
  - puis des contraintes
    - si un ami n'as pas des mésentente, on ajoute la contrainte xi =1 ;
    - si deux amis i et j ne s'entendent pas, xi+xj ≤1
- Montrons que dans un groupe de 6 personnes, soit 3 personnes qui se connaissent mutuellement , soit trois personnes qui ne se connaissent pas.

- Pour tout v € V, dg(v) +d!g(v) = n-1
- donc dg(v) ≥ 3 ou d!g(v) ≥ 3
  - supposons que dg(v) ≥ 3
    - soit il existe une arête entre deux des trois sommets voisins de v , alors on a une clique (et trois personnes se connaissent mutuellement)
    - soit il n'y en a pas et trois personnes ne se connaissent pas mutuellement
  - supposons que d!g(v) ≥ 3
    - idem
- revenons au problème du stable.
- Proposons une heuristique pour résoudre le problème du stable
  - heuristique : ajouter au fur et à mesure un sommet de plus petit degré à l'ensemble stable et tel que ce sommet n'ai pas de voisin dans le stable.
  - **stable(G)**
    - **si G ne contient pas de sommet alors**
      - **retourner Ø ;**
    - **sinon**
      - **choisir un sommet v de plus petit degré dans G**
      - **retourner {v}U stable(G\ N[v] //v et ses voisins) ;**
  - **fin**
- Étant donné un graphe G= (V,E) , un ensemble S est maximal par inclusion si pour out v €V \S , S U {v} n'est pas stable.
  - StableMaximal(G)
    - pour chaque sommet v faire // O(n)
      - marquer[v] ← faux
    - s← Ø
    - pour chaque sommet v faire // O(n+m)
      - si marquer[v] = faux alors
        - marquer v
        - s ← S U {v}
        - pour chaque voisin u de v faire
          - marquer u // somme(d(v)) = 2*m
        - fin pour
      - fin si
    - fin pour
  - fin
- Question 7 : on aimerait bien avoir une famille de ayant un nombre exponentiel de stables maximum .
  - En prenant un graphe avec n composantes connexes avec des cliques de taille 3 ., on Obtient 3^n/3 possibilités
- Construire un algorithme qui va construire tous les stables maximaux d'un graphe.
- Ensuite nous analyserons son temps d'exécution au pire des cas
- Nécessairement le nombre de stables maximaux produits pas l'algorithme sera inférieur ou égal à son temps d'exécution.

- Prendre un sommet v quelconque de plus petit degré dans le graphe
- soit v € stable
- construire tous les stables qui contiennent v
  - c'est à dire, il faut faire un appel récursif sur G – N[v]
- si v n'appartient pas au stable
  - a cause le la maximalité de l'algorithme , au moins l'un de ses voisins appartient au stable
    - pour chaque voisin u de v , construire tous les stables contenant u
      - c'est a dire, ...appel récursif sur G-N[u].
- notons T(n) le temps d'exécution de l'algorithme sur un graphe a n sommets.
  - T(n) = T(n-1-d(v) + ∑(u € N[v]) (n-1 – d(u)).