Mickel Montassier
mickael.montassier@lirmm.fr

HLIN305 - Théorie des graphes:

Un control continu en Amphi le 28 Novembre durée: 1h sans documents
Un control en TP le 12 décembre sur les créneaux de TD et TP

Chapitre 1: Définition de base et connexité

I) Graphes:

Un graphe G(V, E) est constitué

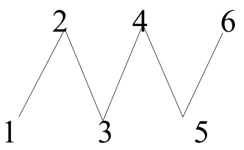
- d'un ensemble (fini) de sommet V, de taille n;
- d'un ensemble d'arêtes E (fini), constitué de paires d'élements de V, de taille m

Deux sommets x, y e V tels que {x, y} e E, sont voisins, reliés, adjacents.

On écrit xy e E, yx e E.

Exemples:

1)

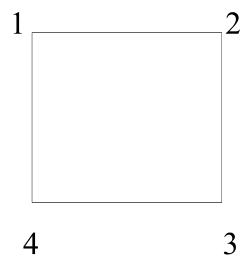


V = {1, 2, 3, 4, 5, 6}
E = {12, 23, 34, 45, 56}

Le chemin (en chaine)
de longueur 5 On le not P6

Plus généralement On note Pk le chemin sur k sommets

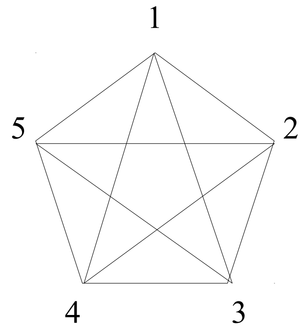
2)



V= {1, 2, 3, 4}
E = {12, 23, 34, 41}

Le cycle de longueur 4 On le note C4
On note Ck le cycle sur k sommets

3)



V= {1, 2, 3, 4, 5}
E = {12, 13, 14, 15, 23, 24, 25, 34, 35, 45}

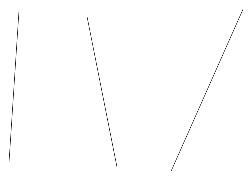
Le graphe complet sur 5 sommets on le note K5

4)

*
*   *
*

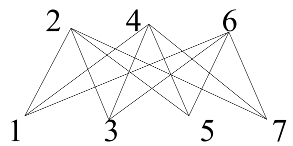
Stable de taille 4 (plus d'arêtes)

5)



Couplage, ensemble d'arêtes deux à deux disjointes.

6)



Ka,b le graphe bipartie complet avec bipartition V= AUB

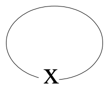
et |A|=a |B|=b

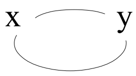
Remarques
-Un graphe est biparti s'il admet une partition de ses sommets en 2 stables

(C6 est biparti, C5 ne l'est pas)

- Soit contraire on interdit les boucles (xx) et les arêtes multiples (plusieurs je is xy)

On ne veut pas





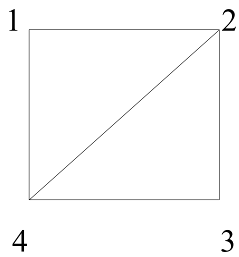
II) Codage

- On code généralement V par {1, ..., n} ou {0, ..., n-1}
- On peut coder les adjacences de 3 façons (il y en a plus!)

1) par liste des arêtes
2) par listes des voisins : Pour tout v e V, on associe L(v) sa liste
3) par matrice d'adjacences: Ag=(aij)(1<=i, j<=n) où aij = 1 si ij e E 0 sinon

Exemple:

G=



V= {1, 2, 3, 4}

1) On code G par {12, 23, 24, 34, 41}

2) On code L(1) = {2, 4}, L(2) = {1, 3, 4}, L(3) = {2, 4}, L(4) = {1, 2, 3}

3) Ag = $\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 0 & 1 \\ 2 & 1 & 0 & 1 & 1 \\ 3 & 0 & 1 & 0 & 1 \\ 4 & 1 & 1 & 1 & 0 \end{array}$

Propriété sur Ag:

1) Ag est symétrique car G est non orienté (xy e E <=> yx e E)

2) 0 sur la diagonale car pas de boucles

Avantages: (m = nb arêtes, n = nb sommets)

	Listes d'arrêtes	Listes de voisins	Matrices d'adjacents
Taille de codage	O(m)	O(n + m)	$O(n^2)$
Temps par selection: "décider xy e E"	O(m) (ou O(log n) si trié)	O(n) (ou O(log n) si trié)	O(1) (temps constant, on suppose l'accés mémoire constant)
Trouver les voisins de x e V	O(m)	O(|L(x)|)	O(n)

Remarques:

- On néglige la taille du codage des entiers: pour coder {1, ..., n}, on prend O(n) et pas O(n log n)
- m <= (n (n – 1)) / 2 => m = $O(n^2)$

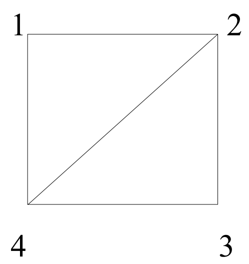
III) Degrés:

Le degré d'un sommet v e V est le nombre de ses voisins, on le note dG(v)

c'est également le nombre d'arêtes incidentes à v (ie. Dont une extrémité est v)

Exemples:

n = 4, m = 5



dG(1) = 2
dG(2) = 3
dG(3) = 2
dG(4) = 3

dG(1) + dG(2) + dG(3) + dG(4) = 10

Formule des degré: Pour tout G = (V, E)

Somme pour tout v e V de dG(v) = 2 |E|
(La somme des degrées de chaque sommets = 2.|E|)

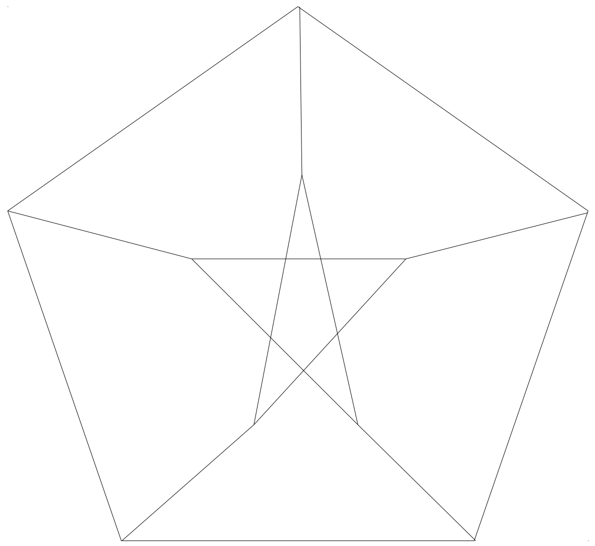
Preuve: Chaque arêtes xy est compilée deux fois, 1 fois pour x, et une autre pour y

Remarques:

- La somme des degré est paire
- Un graphe est k-régulier si tous ses sommets sont de degré exactement k

0-régulier <=> stable
1-régulier <=> couplage
2-régulier <=> Union disjointe de cycles
3-régulier <=> Graphe cubique, structurelement plus compliqué

Le graphe de Peterson:



- Plus petits smark
(X' = Delta + 1)

- Plus petit cubique non hamiltonnien

- Plus grand graphe graphe de Moore pour Delta = 3 et disymétrie 2

IV) Connexité:

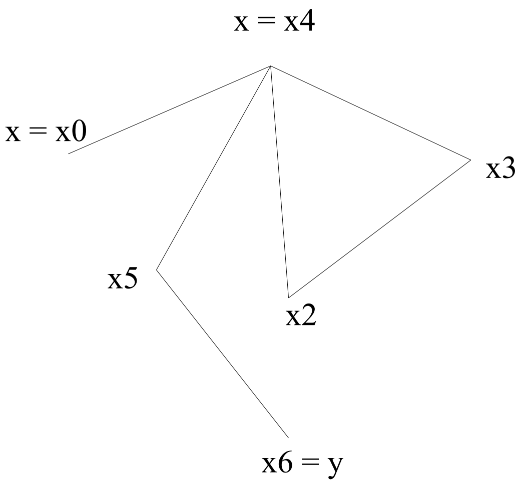
-Soient x,y e V

Une marche de x à y est une suite x = x0, x1, x2,..., xp = y
telle que xi, xi+1 e E (Pour tout i = 0, p – 1)
Sa longueur est l

si tout les sommets sont distincts, alors on a un chemin de x a y

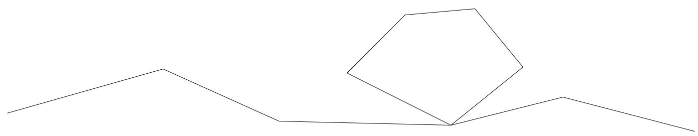
Lemne:

Soient G un graphe. x, y deux sommets de G, avec G contenant une marche de x à y <=> G contient un chemin de x à y.



Preuve:

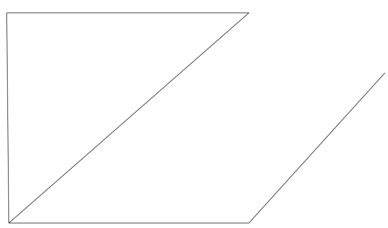
(<=) trivial
(=>) Prenons M une marche de x à y On applique l'algo suivant:
Tant que M contient 2 sommets identiques $x_i = x_j$ avec $i < j$
M <= x_1 x_2 x_{i-1} x_i x_{j+i}...x_p



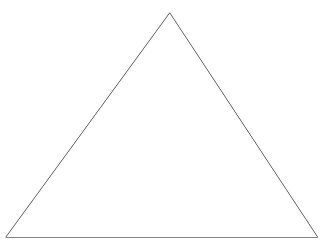
La longueur de M décroit à chaque tour: l'algo s'arrête. On a alors M un chemin

G est connexe si pour tout x, y e V, il existe un chemin de x à y (on peut remplacer chemin par marche) Comment décrire un graphe s'il n'est pas connexe ? Une composante connexe est un ensemble maximal pour l'inclusion de sommets qui forme un graphe connexe.

Exemple:



graphe connexe 1 seule composante connexe



3 composante connexe

Propriété:
Les composantes connexes sont des ensembles disjoints de sommet

Remarques:
Ce sont les classes d'équivalences de la relation x ~ y (si il existe un chemin de x à y)

Connexes <=> 1 seule composante

Pas d'arêtes entre 2 composante connexes distinctes

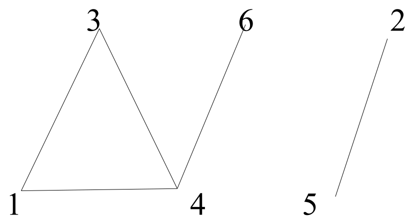
V) Calcul des composantes connexes du graphe:

Entrée: G = (V, E) donné par liste d'arrêtes.
Sortie: comp: V -> V : fonction (ou un tableau) telle que comp(x) = comp(y) ssi x et y appartiennent à la même composante connexe. (en d'autres termes il existe un xy-chemin entre x et y.)

les composantes connexes sont alors les sommets ayant la même valeur de comp.

Exemple:

G = (V, E) V = {1, 2, 3, 4, 5, 6, 7} E = {13, 14, 25, 34, 46}



Principe:

Au début, pour tout x e V, comp(x) = x
("chaque sommet est dans sa propre composante")
On lit les arêtes une à une. A chaque arêtes xy e E, on fusionne les composantes numérotées comp(x) et comp(y).

Arêtes lues	Aucune	13	14	25	34	46
fonction	Comp(1) = 1	Comp(1) = 3	Comp(1) = 3	Comp(1) = 3	Comp(1) = 4	Comp(1) = 6
comp	Comp(2) = 2	Comp(2) = 2	Comp(2) = 2	Comp(2) = 5	Comp(2) = 5	Comp(2) = 5
	Comp(3) = 3	Comp(3) = 3	Comp(3) = 3	Comp(3) = 3	Comp(3) = 4	Comp(3) = 6
	Comp(4) = 4	Comp(4) = 4	Comp(4) = 4	Comp(4) = 4	Comp(4) = 4	Comp(4) = 6
	Comp(5) = 5	Comp(5) = 5	Comp(5) = 5	Comp(5) = 5	Comp(5) = 5	Comp(5) = 5
	Comp(6) = 6	Comp(6) = 6	Comp(6) = 6	Comp(6) = 6	Comp(6) = 6	Comp(6) = 6

(Voir algo poly cours)

Analyse de Composante:

- Terminaison:
L'algo contient que des "boucles pour" et des tests. Il termine.

- Preuve:
(voir preuve poly de cours)

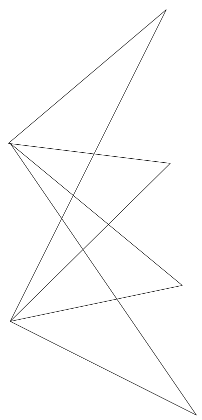
Problème du jour n°1:

Cherchons des marches ou chemins ayant diverses prorpiétés. Regardons la proriété "passons par tous les sommets".

Probleme 1: Existe t-il une marche qui passe par tous les sommets ?
Réponse: Oui ssi le graphe est connexe et on sait faire !

Problème 2: Existe t-il un chemin qui passe par tous les sommets ? (CHEMIN HAMILTONIEN)

K2,4:

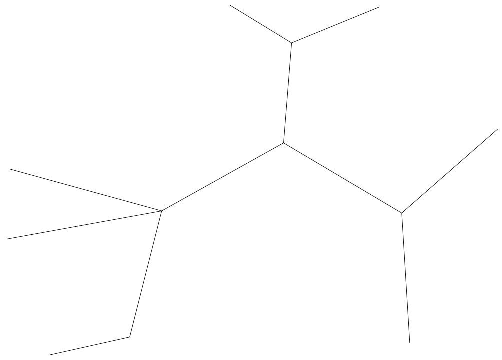


Réponse: Pas toujours, Ex: K2, 4, Nous n'avons à ce jour pas de "bons" algorithmes (polynomial) qui permet de décider si un tel chemin existe Actuellement que des algo exponentielle.

Chaptitre 2: Arbres couvrants:

I) Arbres:

Exemples:



Un arbre: T = (V, A) est un graphe connexe sans cycles.
Une feuille de T est un sommet de degré 1.

Un arbre a-t-il toujours une feuille ?

Proposition:

Tout arbre ayant au moins 2 sommets a au moins 2 feuilles.

Preuve: Soient v1, ..., vt un chemin maximal (inextensible)

v1 — v2 — ... — vt-1 — vt

v1 n'a pas d'autre voisin que v2 sinon on peut étendre le chemin ou créer un cycle. D'où v1 est de degré 1 v1 est une feuille.
Idem pour vt

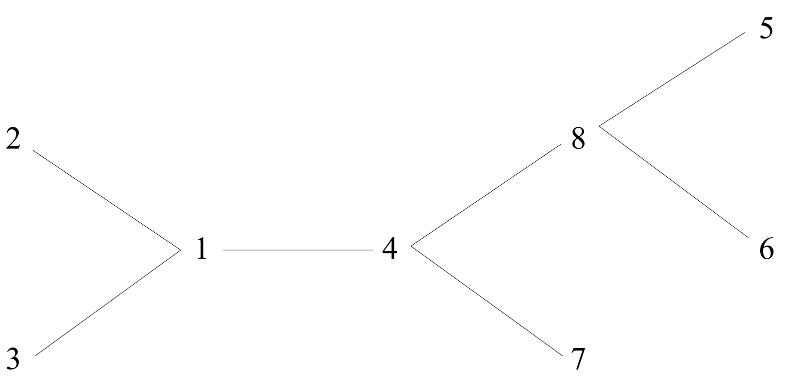
Remarques:

Si f est une feuille de T, T\{f} (T privé de f) est encore un arbre.
Peut constituer:
Si T est un arbre à n sommets, on note f1 une feuille de T

f1 ——— T\{f1}
f2 ——— T\{f1, f2}
...

On obtient ainsi une énumération des sommets de T: f1, f2, ..., fn-1, fn appelée schéma d'élimination de T.
Pout tout i, fi est une feuille deu sous-arbre de T réduit à {f1, ... fi}

Exemple:

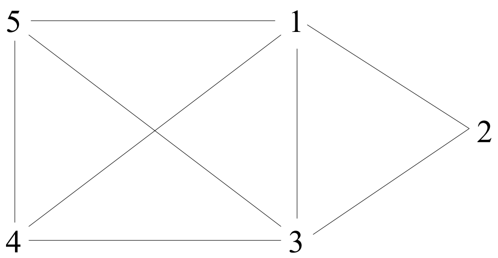


1, 2, 4, 8, 3, 7, 6, 5

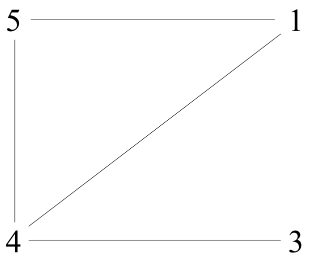
II) Arbres couvrants:

Soit G = (V, E) un graphe. Un graphe H = (V', E') avec V'cV
et E' c E est un sous-graphe de G. Si de plus V' = V, alors H est un sous-graphe couvrant.

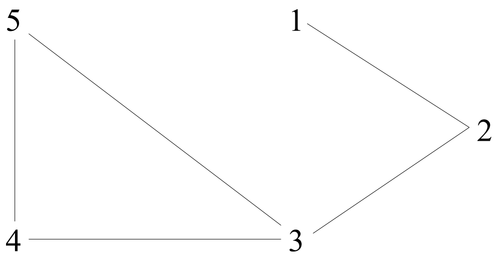
Exemple:



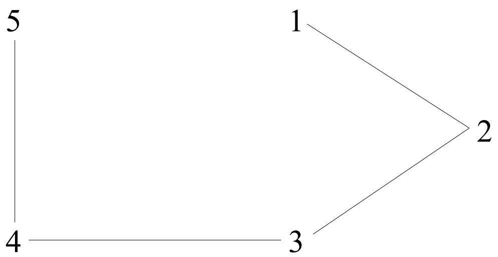
(G)



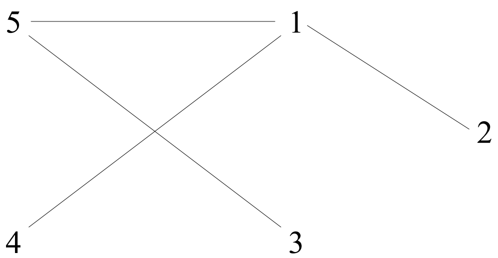
(Sous graphe de G)



(Sous graphe de G couvrant (mais pas arbre))



('arbres couvrants)



(2eme ex d'arbres couvrants)

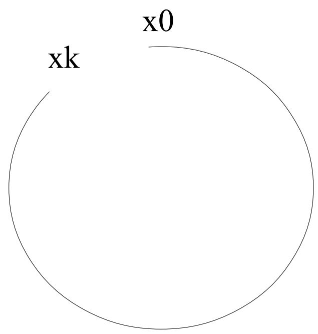
Proposition:
Un graphe G est connexe ssi il admet un arbre T couvrant.

Preuve:

(<=) Immédiat, un xy-chemin de T est un xy-chemin de G
(=>) Supposons G connexe, Si G n'a pas de cycles alors G est un arbre

Supposons G possède un cycle x0, x1, ... xk, x0

Si on supprime l'arrête x0, x1, le graphe reste connexe: toute marche empruntant x0x1 peut être re-router en passant par x0xkxk-1...x2x1



On applique l'algo suivant:

Tant que il existe un cycle C de G { faire Enlever une arête de C }
L'algo termine: le nombre de cycles (en arêtes) décroit strictement à chaque itération

À la fin de l'algo, on obtient:

1) un sous-graphe couvrant (on a supprimé que des arêtes)
2) Ce sous-graphe ne contient pas de cycles (condition de continuation)

On obtient donc un arbre couvrant

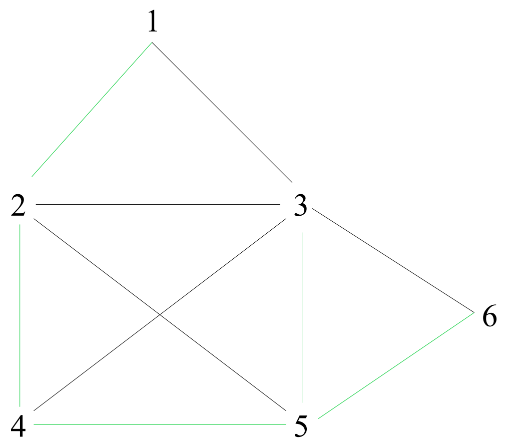
Remarque:

Cet algo calcule un arbre couvrant, On verra plus rapide. Un arbre couvrant est un certificat de connexité.

II) Un algorithme de calcul d'un arbre couvrant:

(voir poly)

Exemple:



E = {24, 35, 56, 36, 45, 32, 12, 13}

A? 24, 35, 56, 45, 12

Analyse de l'Arbre couvrant:

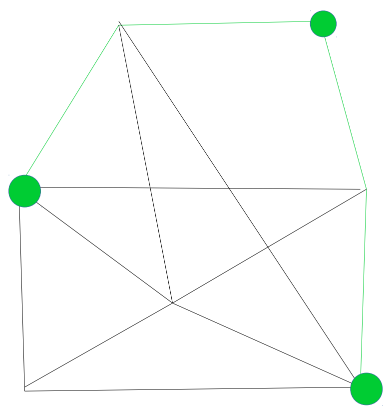
(voir poly)

Problème du jour N°2:

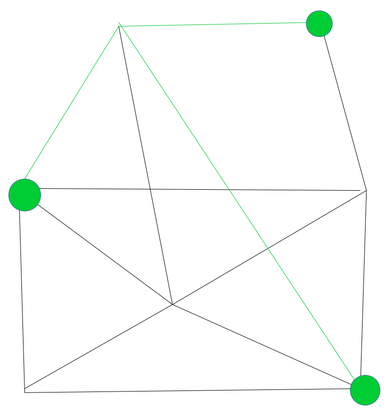
Problème (Stainer tree):

Entrée: G = (V, E) connexe, S c= V, p e N
Sortie: T = (V, A) un sous arbre de G couvrant s avec |A| <= p

"Connecter les sommets de s à un cout minimum"



Coût: 4



Coût: 3

Le problème est dit Np-difficile, "il s'agit de l'équivalent de NP-complet des problemes de décision vaux problèmes d'optimisation (i.e. trouver la meilleur solution). Actuellement, on ne connait pas d'algorithme polynomial, ("que" des exponentiel).

<u>Remarques:</u>

- Si S = V, on sait faire c'est arbre couvrant.
- Si S = {x, y}, cela revient à chercher un plus court chemin entre x et y (chapitre 4)

<u>V Quelques propriétés;</u>

<u>Proposition:</u> un arbre sur n sommets contient n-1 arêtes
<u>Preuve:</u> Induction:
- Vrai si n = 1.
- Supposons que la propriété est vrai sur tous les arbres n sommets et considérons T un arbre à n + 1 sommets

T possède une feuille f. T\f est un arbre sur n sommets. Par induction, T\f possède n-1 arêtes ainsi T possède (n-1) + 1 = n arêtes

Une forêt est une union disjointe d'arbres.

Proposition: Un graphe est une forêt ssi, il ne possède pas de cycles.
Preuve:
(=>) Immédiat.
(<=) Si G est sans cycle, chacune de ses composantes connexes est sans cycles et connexe, donc un arbre

Proposition: Une forêt à n sommets ayant c composante connexes possède n – c arêtes.

Preuve:
On note n1, n2, ..., nc la taille de chacune des composantes connexes.
On observe: n1 + n2 + ... + nc = n

m = (n1 – 1) + (n2 – 1) + ... + (nc -1) = n – c

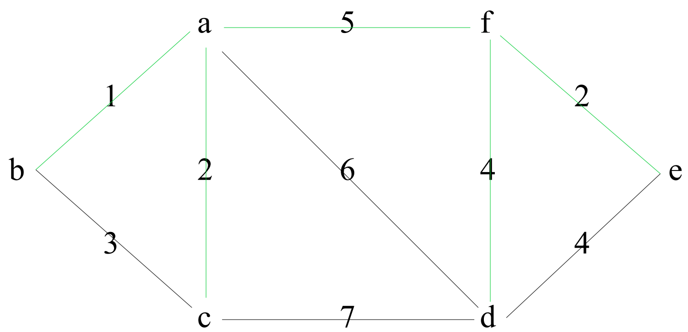
VI) Arbres couvrant de poids minimum:

Problème: ACPM
Entrée: G = (V, E) connexe et w : E -> R^t une fonction de poids sur les arêtes.
Sortie: T = (V, A) un arbre couvrant de G tel que w(A) = Somme de e eE w(e) soit minimum

Objectif: Relier tous les sommets à coût minimum.

(voir algo Kruskal).

Exemple:



E trié = {ba, fe, ca, ~~cb~~, df, ~~de~~, af, ~~da~~, ~~cd~~}

Remarques:

La solution par l'algo Kruskal n'est pas unique: cela dépend de l'odre des arêtes de poids egal.
Algo glouton. (pas backtrack)
En randomisé: O(m)
En non-randomisé: O(m.a(n, m)) => a = niveau de fonction d'ackerman

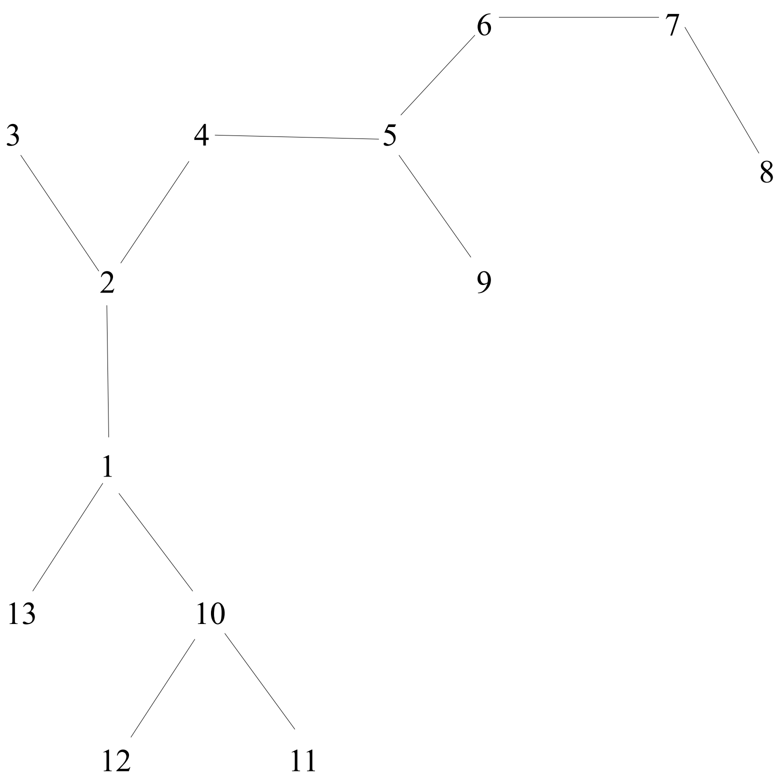
VII) Problème du jeu n°3:

TSP:
Problème: (Voyageur de commercer)
Entrée: Un ensemble n points du points du plan (1, ..., n)
Sortie: une énumération des sommets (v1, ... , vn telle que d(v1, v2) + d(v2, v3) + ... + d(vn-1, vn) soit minimum.

C'est un problème NP-difficile. Dans TSP, il est possible d'approcher la solution optimale à facteur 3 en temps polynomial: On parle d'une 2-approximation.

2-approx:

-Construire Kn sur {1, ..., n} pondéré par w(ij) = d(i, j)
-On calcule T ACPM pour le graphe ci-dessus
-On parcours T de façon à traverser chaque arête exactement 2 fois.
-Modifier le parcours de façon à ne passer qu'une fois par chaque sommet (raccourcir)



Remarques:

- Christofides (1976), 3/2-approx dans le cas métrique (inégalité triangulaire)
- Sebö, Vygen (2012), 7/5-approx dasn le casa graphique

// ATTENTION //
COURS A INSERER ICI
//

Chapitre 4: Graphes Orientés:

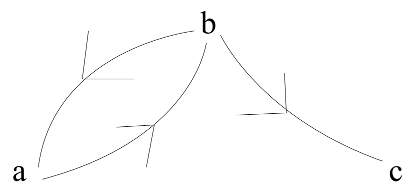
1) Définitions:

Un graphe orienté D=(V, A) est formé de sommets V et d'arêtes A qui sont des couples d'éléments distincts de V (et non plus des paires).

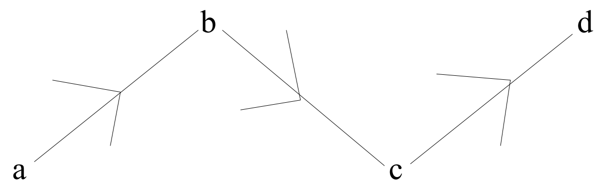
Si xy est un acr du D, alors yx peut l'être aussi mais pas nécessairement.

Exemples:

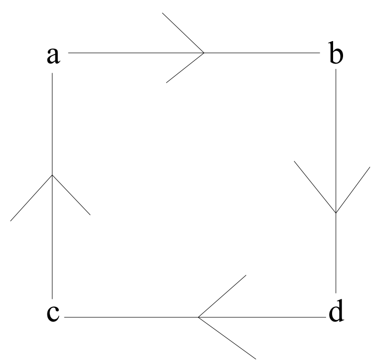
1)



2)

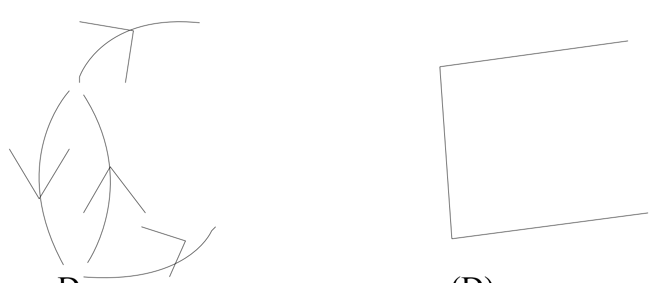


3)



lorsque xy e A, on dit que y est un voisin sortant de x et est un voisin entrant de y. Le voisinage sortant de x: N+(x)={y : xy e A} ou vois+(x)

le degrés sortant de x est dt(x) = |N+(x)|
Idem pour le voisinage entrant N- ou vois- et le degré entrant d-(x)



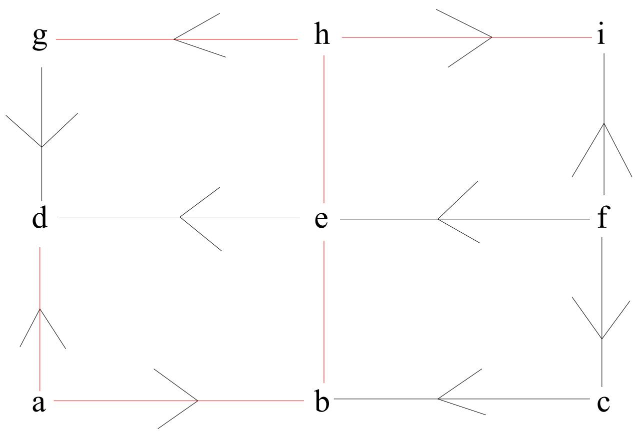
D u(D)

lorsque d-(x) = 0, alors x est une source. Si d+(x) = 0, alors x est un puit, lorsqu'on oublie les orientations, on obtient un graphe non orienté, dit graphe sous-jacent, noté u(D)

Codage: On peut coder par:
- liste des arcs.
- listes des voisins sortants pour chaque x e V: N+(x)
- matrice d'adjacences. Où aij = 0 si ij non(e) A, 1 si ij e A
AD = (aij) 1<= ij <= n (AD n'est pas nécesserement symétrique).

2) Parcours:

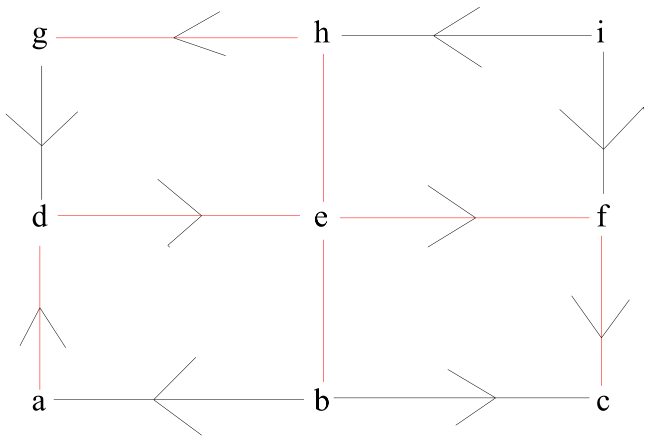
Identique au cas non orienté en exeminant vois+(x) au lieu de vois(x)
On part d'une racine et on suit les arcs sortants. On va obtenir des arborescences sortantes. Il s'agit de l'orientation d'un arbre enraciné en r tq: pour tout x, il existe chemin orienté de r à x.



Parcours en largeurs:

AT: a, b, d, e, h, g, i

Sommet	a	b	c	d	e	f	g	h	i
pere	a	a	rien	a	b	rien	h	e	h
ordre	1	2	rien	3	4	rien	6	5	7
niveau	0	1	rien	1	2	rien	4	3	4

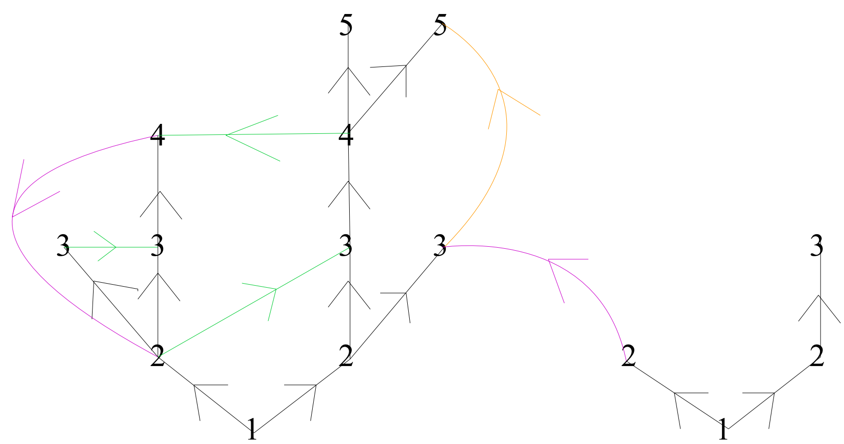


Parcours en profondeurs:

AT: g, d, e, b, a

Sommet	a	b	c	d	e	f	g	h	i
pere	b	e		g	d	e	g	e	f
ordre	5	4		2	3	8	1	12	9
niveau	6	7		15	14	11	16	13	10

Parcours en largeurs:

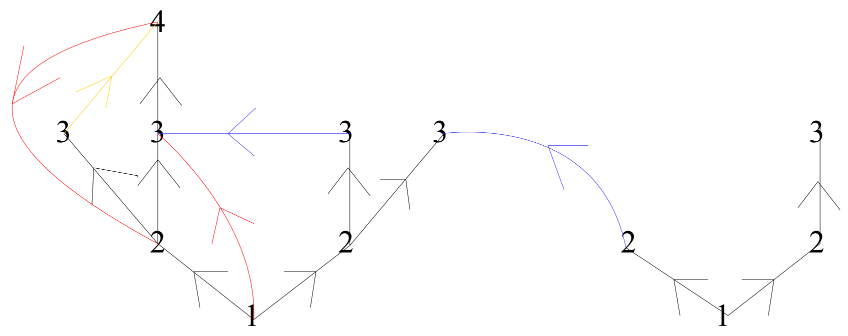


- Entre niveau identique: posible
- D'un niveau k à k+1 toujours possible.
- D'un niveau, k à k' avec

  k < k'+ 1 impossible

  k > k' possible nouveau

Parcours en profondeurs:



- À l'interieur d'une même branche: posible
- D'un sommet x vers un sommet y d'une autre branche
  - si $d(x) > d(y)$ possible nouveau
  - si $d(x) < d(y)$ impossible

Exemple:

2

1

2 4

1

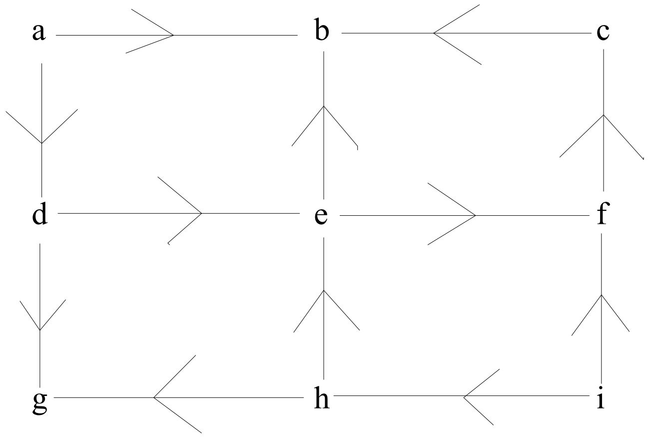
3

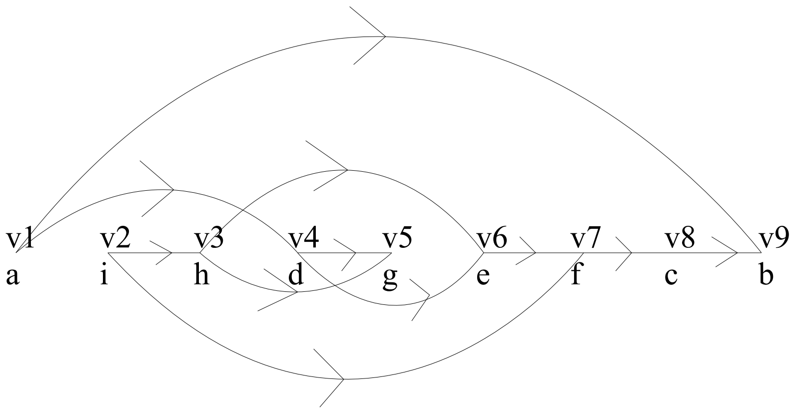
3 5

1 2

Le sommet xk n'a pas de voisin entrant à l'extérieur de x1, ... xk-1, sinon le chemin ne serait pas maximal. Si il existe xi, 1 <= i <= k-i, voisins entrant de xk, alors on a un circuit, impossible.

D'où xk est une source. De même, tout DAG possède un puit.
Un tritopologique d'un graphe orienté est une énumeration v1, v2, ..., vn des sommets tq pour tout xixj, e A, i < j (les arcs sont vont de gauches à droite)





<u>Proposition:</u>

D est un DAG ssi il admet un tritopologique
(<=) Si D admet un tritopologique il ne peut pas avoir de circuits.
(=>) si D est un DAG, il a une source x1 D-x1 est un DAG, il possède une source x2. On réitère l'énumération x1, x2, ... xi est un tritopologique

<u>4) Problème du jour n°6:</u>

<u>Problème:</u> (2-chemins-disjoints(D, {x1, x2},{y1, y2}))
<u>Entrée:</u> D graphe orienté, x1, x2 departs, y1, y2, arrivées.
<u>Sorties:</u> 2 cheminsdisjoints partans de {x1, x2] arrivant en {y1, y2}
Il existe un algo polynomial (flots)

"l'obstruction: 1 sommets separtateurs"

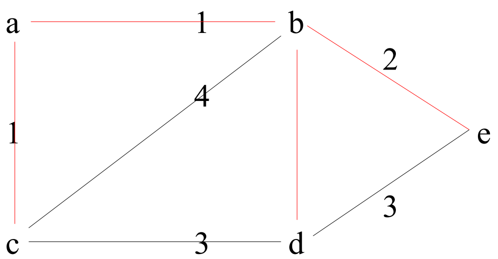
Si on spécifie x1 y1
x2 y2

Chapitre 5: Plus court chemins

1) Graphes valués:

On cherche un arbre des plus courts chemins, mais maintenant il y a des longueurs sur les arêtes.

Problème: ARBRE DES PLUS COURS CHEMINS
Entrées: G = (V, E), P: E -> R+, r e V
Sortie: Un arbre des plus courts chemins issus de r.



Pour chaque sommet x, le chemin ldans l'arbre de x à r est un PCC de x à r.

Algorithmes manuel:

- Clouer au mur la racine
- Remplacer les sommets par des perles
- Remplacer xy par un fil de longueur y
- Lachez tout

2) Algorithem Dijkstra (1959)

Edsgar Wybe Dijkstra (1939-2002), mathématicien informamticien néerlandais, Prix Turing 1972.
1968: A case against the GOTO statement.
-> vers la programmation structurée.

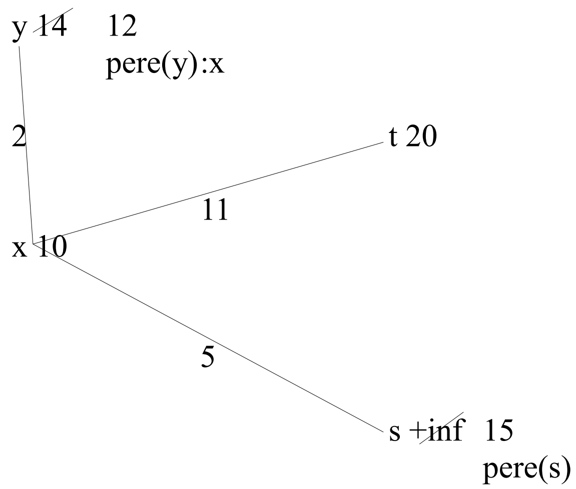
Pour chaque somet x, on calcul d(x) la distance de r à x.
On maintient un ensemble de sommet traités.
A chaque étape, on traite le sommet non encore traité le plus proche de r.

Quand on choisit un sommet, on texte s'il forme un raccourci pour atteindre ses voisins nontraités.
Si oui, on met à jour les distance correspondantes.
On dit que l'on "relaxe" l'arête.

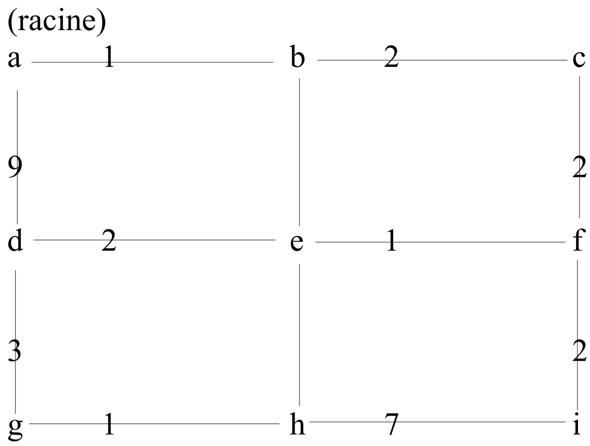
Traités

Non traités

r



z 15

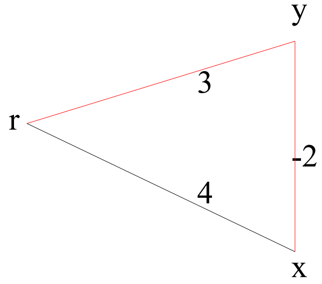


Sommet	a	b	c	d	e	f	g	h	i
Pere	a	a	b	~~a~~ e	~~b~~ f	c	d	~~e~~ ~~i~~ g	f
d	0	1	3	~~9~~ 8	~~8~~ 6	5	11	~~16~~ ~~14~~ 12	7

<u>Remarques:</u>

- On obtient un arbre des plus courts chemins issus de r
- Idem en orienté en remplaçant Vois(x) par Vois+(x) Arboressence sortante des plus courts chemins issus de r

s'il y a des poids négatifs (la ne marche plus)

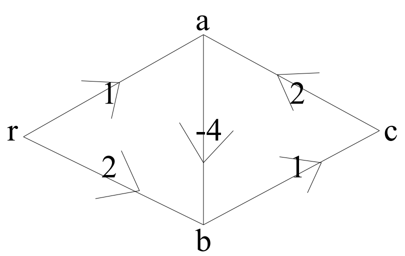


Cependant r xy amène en y en 2

3) Graphes orientés et poids négatifs sur les arcs:

On va étendre la résolution du problème en cas où il n'y a pas de circuits de poids négatif.

Exemple:



f(r ab c) = -2
f(r abc ab c) = -3
f(r (ab)^h) = -1 – h

non existence d'une plus courte marche de r à c