

Théorie des Graphes

Jean-Charles Régin, Arnaud Malapert

Théorie des Graphes - 2015/2016

Remerciements

2

- Didier MAQUIN (voir son site web)
- Didier MULLER

Théorie des Graphes - 2015/2016

Livres

3

- Claude Berge : « Graphes »
- Michel Gondran et Michel Minoux : « Graphes et Algorithmes »
- Christian Prins : « Algorithmes de Graphes »
- Eugene Lawler : « Combinatorial Optimization »
- Ahuja, Magnanti et Orlin « Network Flows »
- R. Tarjan : « Data Structures and Network Algorithms »

Théorie des Graphes - 2015/2016

Introduction

Théorie des Graphes - 2015/2016

Graphes : 3 aspects

5

- Communiquer / Visualiser
- Définir des problèmes
- Raisonner

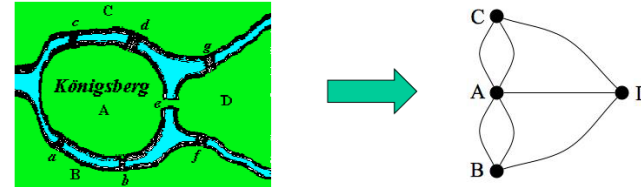
- Pas que des algos : aussi des problèmes

Théorie des Graphes - 2015/2016

Histoire

6

Naissance en 1736, communication d'Euler où il propose une solution au problème des ponts de Königsberg



Dessin comportant des **sommets** (points) et des **arêtes** reliant ces sommets

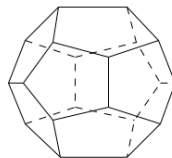


Théorie des Graphes - 2015/2016

Histoire

7

- 1847 Kirchhoff
théorie des arbres (analyse de circuits électriques)
- 1860 Cayley
énumération des isomères saturés des hydrocarbures C_nH_{2n+2}
- A la même époque, énoncé de problèmes importants
 - Conjecture des quatre couleurs (1879)
(Möbius, De Morgan, Cayley, solution trouvée en 1976)
 - Existence de chemins Hamiltoniens (1859)
- 1936 König
premier ouvrage sur les graphes
- 1946 ➡ Kuhn, Ford, Fulkerson, Roy
Berge



Théorie des Graphes - 2015/2016

Définitions

Théorie des Graphes - 2015/2016

Graphe : définitions

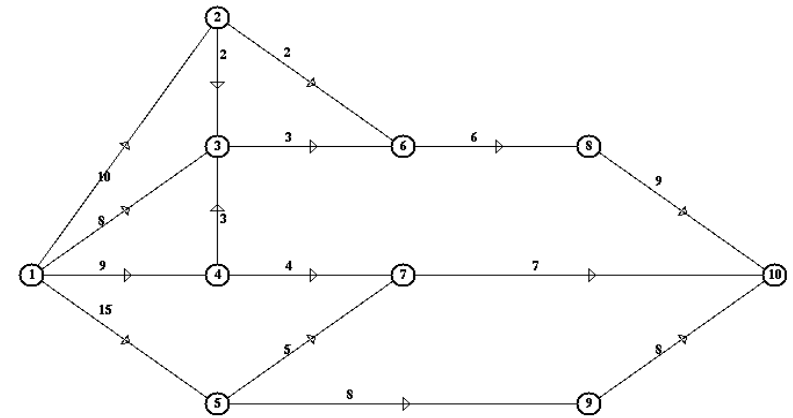
9

- Un Graphe Orienté (ou Digraph (directed graph)) $G=(X,U)$ est déterminé par la donnée :
 - ▣ d'un ensemble de **sommets** ou **nœuds** X
 - ▣ d'un ensemble ordonné U de couples de sommets appelés **arcs**.
- Le nombre de sommets d'un arc est l'ordre du graphe
- Si $u=(a,b)$ est un arc de G alors
 - ▣ a est l'extrémité initiale de u
 - ▣ b l'extrémité terminale de u .
 - ▣ b est le successeur de a
 - ▣ a et b sont adjacents
- Les arcs **ont un sens**. L'arc $u=(a,b)$ va de a vers b .
- Ils peuvent être munit d'un coût, d'une capacité etc...

Théorie des Graphes - 2015/2016

Graphe

10



Théorie des Graphes - 2015/2016

Graphe

11

- On note par $\omega(i)$: l'ensemble des arcs ayant i comme extrémité
- On note par $\omega^+(i)$: l'ensemble des arcs ayant i comme extrémité initiale = ensemble des arcs sortant de i
- On note par $\omega^-(i)$: l'ensemble des arcs ayant i comme extrémité terminale = ensemble des arcs entrant dans i
- $\Gamma(i)$: ensemble des successeurs de i
- $d^+(i)$: degré sortant de i (nombre de successeurs)
- $d^-(i)$: degré entrant de i (nombre de sommets pour lesquels i est un successeur)

Théorie des Graphes - 2015/2016

Graphe non orienté

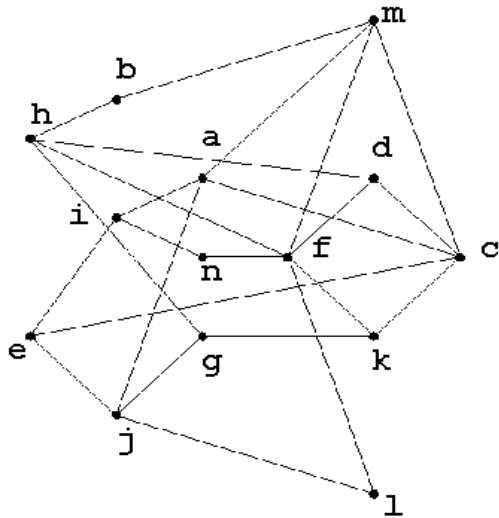
12

- Un Graphe non orienté $G=(X,E)$ est déterminé par la donnée :
 - ▣ d'un ensemble de sommets ou nœuds X
 - ▣ D'un ensemble E de paires de sommets appelées **arêtes**
- Les arêtes ne sont pas orientées

Théorie des Graphes - 2015/2016

Graphe non orienté

13



Théorie des Graphes - 2015/2016

Graphe : définitions

14

- Deux sommets sont voisins s'ils sont reliés par un arc ou une arête
- $N(i)$: ensemble des voisins de i : ensemble des sommets j tels qu'il existe une arête contenant i et j

Théorie des Graphes - 2015/2016

Graphe complet

15

- Un graphe $G=(X, A)$ est dit **complet** si, pour toute paire de sommets (x, y) , il existe au moins un arc de la forme (x, y) ou (y, x) .
- Un graphe simple complet d'ordre n est noté K_n . Un sous-ensemble de sommets $C \subset X$ tel que deux sommets quelconques de C sont reliés par une arête est appelé une **clique**.

Théorie des Graphes - 2015/2016

Exercices

16

- Reliez le nombre d'arêtes et les degrés
- Montrez qu'un graphe simple a un nombre pair de sommets de degré impair
- Montrez que dans une assemblée de n personnes, il y en a toujours au moins 2 qui ont le même nombre d'amis présents
- Est-il possible de relier 15 ordinateurs de sorte que chaque appareil soit relié exactement à 3 autres ?

Théorie des Graphes - 2015/2016

Exercices (suite)

17

Essayez d'exprimer (et non nécessairement de résoudre...) en termes de graphes les problèmes suivants :

- Peut-on placer quatre dames sur un échiquier 4x4 sans qu'aucune d'elles ne puisse en prendre une autre ?
- Un cavalier peut-il se déplacer sur un échiquier en passant sur chacune des cases une fois et une seule ?
- Combien doit-on placer de dames sur un échiquier 5x5 afin de contrôler toutes les cases ?

Théorie des Graphes - 2015/2016

Représentation des graphes en machine

Théorie des Graphes - 2015/2016

Représentation des graphes

19

- 3 types de représentation qui ont chacune leur avantage et inconvénient
 - Liste de successions
 - Matrice d'adjacence
 - Matrice d'incidence

Théorie des Graphes - 2015/2016

Liste de succession

20

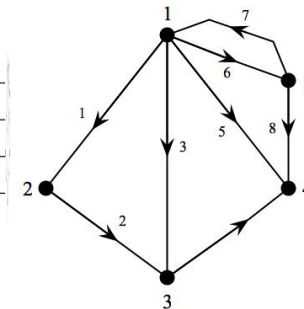
Listes de succession

1	2, 3, 4, 5
2	3
3	4
4	-
5	1, 4

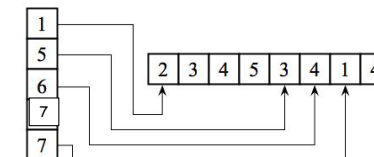
Successeurs

1	5
2	1
3	1, 2
4	1, 3, 5
5	1

Prédécesseurs



Représentation machine



Théorie des Graphes - 2015/2016

Matrice d'adjacence

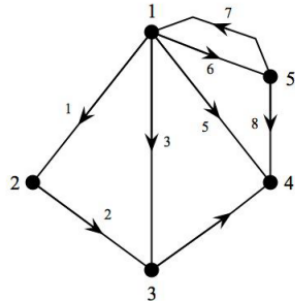
21

Considérons un graphe $G=(X, A)$ comportant n sommets. La **matrice d'adjacence** de G est égale à la matrice $U=(u_{ij})$ de dimension $n \times n$ telle que :

$$u_{ij} = \begin{cases} 1 & \text{si } (i, j) \in A \text{ (c'est-à-dire } (i, j) \text{ est une arête)} \\ 0 & \text{sinon} \end{cases}$$

Une telle matrice, ne contenant que des « 0 » et des « 1 » est appelée **matrice booléenne**.

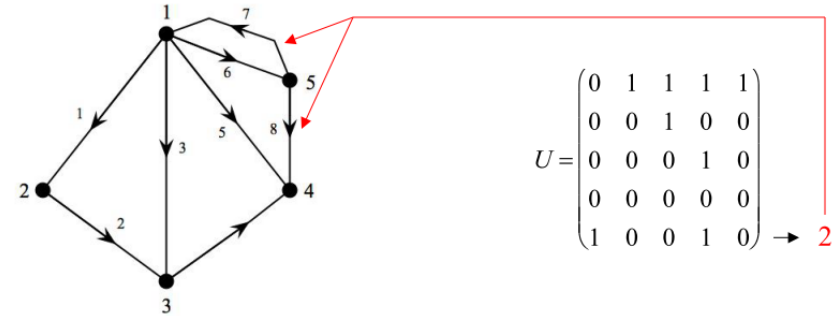
$$U = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$



Théorie des Graphes - 2015/2016

Matrice d'adjacence

22



$$U = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix} \rightarrow 2$$

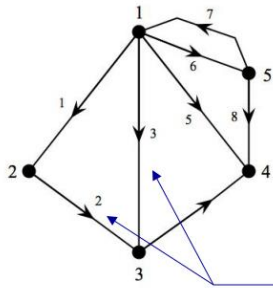
Propriétés

➤ la somme des éléments de la $i^{\text{ème}}$ ligne de U est égale au degré sortant $d_s(x_i)$ du sommet x_i de G .

Théorie des Graphes - 2015/2016

Matrice d'adjacence

23



$$U = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix} \downarrow 2$$

Propriétés

➤ la somme des éléments de la $i^{\text{ème}}$ ligne de U est égale au degré sortant $d_s(x_i)$ du sommet x_i de G .

➤ la somme des éléments de la $j^{\text{ème}}$ colonne de U est égale au degré entrant $d_e(x_j)$ du sommet x_j de G .

➤ U est symétrique si, et seulement si, le graphe G est symétrique.

Théorie des Graphes - 2015/2016

Matrice d'incidence

24

Considérons un graphe orienté sans boucle $G=(X, A)$ comportant n sommets x_1, \dots, x_n et m arêtes a_1, \dots, a_m . On appelle **matrice d'incidence (aux arcs)** de G la matrice $M=(m_{ij})$ de dimension $n \times m$ telle que :

$$m_{ij} = \begin{cases} 1 & \text{si } x_i \text{ est l'extrémité initiale de } a_j \\ -1 & \text{si } x_i \text{ est l'extrémité terminale de } a_j \\ 0 & \text{si } x_i \text{ n'est pas une extrémité de } a_j \end{cases}$$

INVERSE

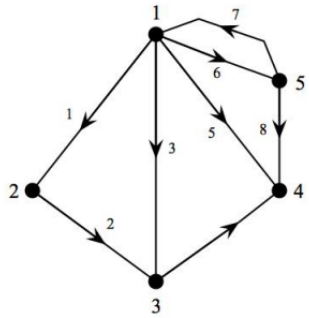
Pour un graphe non orienté sans boucle, la matrice d'incidence (**aux arêtes**) est définie par :

$$m_{ij} = \begin{cases} 1 & \text{si } x_i \text{ est une extrémité de } a_j \\ 0 & \text{sinon} \end{cases}$$

Théorie des Graphes - 2015/2016

Matrice d'incidence

25



Arc 1 2 3 4 5 6 7 8

$$M = \begin{pmatrix} -1 & 0 & -1 & 0 & -1 & -1 & 1 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 \end{pmatrix} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix}$$

Sommet

Théorie des Graphes - 2015/2016

Exercices

26

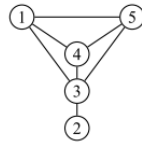
- Pour chacune des représentations, calculer le coût des opérations suivantes
 - ▣ Est-ce que i et j sont reliés ?
 - ▣ Combien i a-t-il de voisins ?
 - ▣ Quels sont les voisins de i ?
 - ▣ Supprimer l'arc (a,b)
 - ▣ Supprimer le premier voisin de i

Théorie des Graphes - 2015/2016

Exercices

27

- Tracer la matrice d'adjacence du graphe



- On a calculé ci-dessous les matrices M^2 et M^3 (M est la matrice ci-dessus). Pour chacune de ces matrices, à quoi correspondent les nombres obtenus ?

$$M^2 = \begin{pmatrix} 3 & 1 & 2 & 2 & 2 \\ 1 & 1 & 0 & 1 & 1 \\ 2 & 0 & 4 & 2 & 2 \\ 2 & 1 & 2 & 3 & 2 \\ 2 & 1 & 2 & 2 & 3 \end{pmatrix} \quad M^3 = \begin{pmatrix} 6 & 2 & 8 & 7 & 7 \\ 2 & 0 & 4 & 2 & 2 \\ 8 & 4 & 6 & 8 & 8 \\ 7 & 2 & 8 & 6 & 7 \\ 7 & 2 & 8 & 7 & 6 \end{pmatrix}$$

Théorie des Graphes - 2015/2016

Chemins, chaînes, cycles et circuit

Théorie des Graphes - 2015/2016

Chaîne

29

- Une **chaîne** est une séquence finie et alternée de sommets et d'arêtes, débutant et finissant par des sommets, telle que chaque arête est incidente avec les sommets qui l'encadre dans la séquence.
- Le premier et le dernier sommet sont appelés (**sommets**) **extrémités** de la chaîne.
- La **longueur** de la chaîne est égale au nombre d'arêtes qui la composent.
- Si aucun des sommets composant la séquence n'apparaît plus d'une fois, la chaîne est dite **chaîne élémentaire**.
- Si aucune des arêtes composant la séquence n'apparaît plus d'une fois, la chaîne est dite **chaîne simple**.

Théorie des Graphes - 2015/2016

Cycle

30

- Un **cycle** est une chaîne dont les extrémités coïncident.
- Un **cycle élémentaire** (tel que l'on ne rencontre pas deux fois le même sommet en le parcourant) est un cycle minimal pour l'inclusion, c'est-à-dire ne contenant strictement aucun autre cycle.

Théorie des Graphes - 2015/2016

Chemin

31

- Un **chemin** est une séquence finie et alternée de sommets et d'arcs, débutant et finissant par des sommets, telle que chaque arc est sortant d'un sommet et incident au sommet suivant dans la la séquence (cela correspond à la notion de chaîne « orientée »).
- Si aucun des sommets composant la séquence n'apparaît plus d'une fois, le chemin est dit **chemin élémentaire**.
- Si aucune des arêtes composant la séquence n'apparaît plus d'une fois, le chemin est dit **chemin simple**.
- Un **circuit** est un chemin dont les extrémités coïncident.

Théorie des Graphes - 2015/2016

Exercice : matrice d'adjacence

32

On considère un graphe $G = (E, U)$ avec $E = \{A, B, C, D, E\}$ et M sa matrice associée.

- Tracer le graphe représentatif de cette matrice.
- Déterminer la matrice d'incidence de ce graphe.
- Calculer M^i , $i \in \{1, 2, 3\}$. Rappeler la signification des coefficients non nuls de ces matrices.
- Calculer $A = I + M + M^2 + M^3 + M^4$. Donner une interprétation de A .

0	1	0	1	0
0	0	1	0	0
0	0	0	0	1
0	0	1	0	1
0	0	0	0	0

Théorie des Graphes - 2015/2016

Exercice : matrice d'adjacence

33

$$M^2 = \begin{pmatrix} 0 & 0 & 2 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad M^3 = \begin{pmatrix} 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad M^4 = 0$$

$$I + M + M^2 + M^3 + M^4 = \begin{pmatrix} 1 & 1 & 2 & 1 & 3 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Théorie des Graphes - 2015/2016

Exercice: jeu des allumettes

34

Deux joueurs disposent de deux tas de trois allumettes, à tour de rôle chaque joueur peut enlever une ou deux allumettes dans un des tas. Le joueur qui retire la dernière allumette perd la partie.

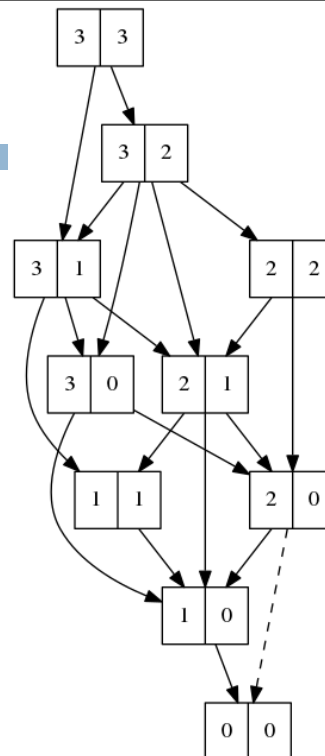
- Modéliser le jeu à l'aide d'un graphe.
- Que doit jouer le premier joueur pour gagner à coup sûr ?

Théorie des Graphes - 2015/2016

Graphe d'état

35

- Chaque état est représenté par un sommet (x,y) indiquant le nombre d'allumettes de chaque tas.
- On élimine les symétries entre les états $(x \geq y)$.
- Il existe un arc entre deux sommets A et B s'il est possible de passer de l'état A à l'état B par une transition valide (enlever une ou deux allumettes dans un des tas).
- Notez que la transition entre $(2,0)$ est $(0,0)$ correspond à un suicide, alors qu'on peut jouer un coup gagnant.

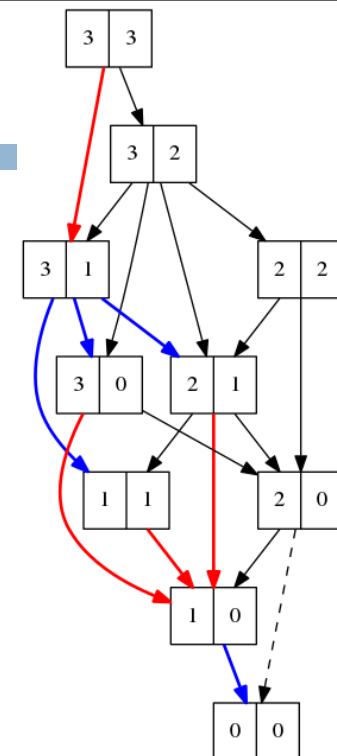


Théorie des Graphes - 2015/2016

Stratégie gagnante

36

- Il faut retirer deux allumettes d'un tas.
- Quel que soit le coup joué par l'adversaire, on peut jouer un coup gagnant.



Théorie des Graphes - 2015/2016

Exercice : Die Hard !

37

- On souhaite prélever 4 litres de liquide dans un tonneau. Pour cela, nous avons à notre disposition deux récipients (non gradués !), l'un de 5 litres, l'autre de 3 litres...
- Comment doit-on faire ?

Théorie des Graphes - 2015/2016

Exercice : Die Hard !

38

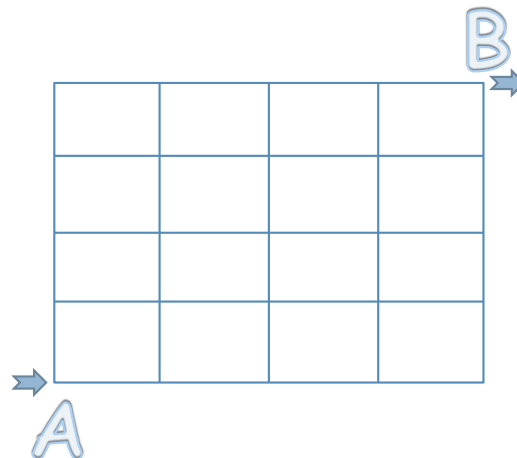
- Chaque état est représenté par un sommet (x,y) indiquant le nombre de litres dans la grande et la petite bouteille.
- Il existe un arc entre deux sommets A et B s'il est possible de passer de l'état A à l'état B par une transition valide :
 - vider une bouteille (V);
 - remplir une bouteille (R);
 - transvaser le contenu de la bouteille A dans la bouteille B jusqu'à ce que la bouteille A soit vide ou la bouteille B soit pleine (T).
- Solution : $(0,0) \rightarrow R \rightarrow (5,0) \rightarrow T \rightarrow (2,3) \rightarrow V \rightarrow (2,0) \rightarrow T \rightarrow (0,2) \rightarrow R \rightarrow (5,2) \rightarrow T \rightarrow (4,3)$

Théorie des Graphes - 2015/2016

Exercice : dénombrement de chemins

39

- Supposons une grille de dimensions $n \times m$ (par ex. 4×4).
- Quelle est la longueur d'un plus court chemin entre A et B ?
- Comment pouvez-vous représenter un tel chemin ?
- Combien y a-t-il de plus courts chemins entre A et B ?

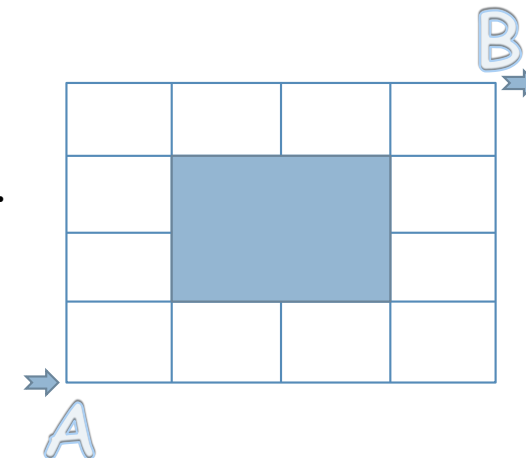


Théorie des Graphes - 2015/2016

Exercice : dénombrement de chemins

40

- Supposons une grille de dimensions $n \times m$ (par ex. 4×4).
- Combien y a-t-il de plus courts chemins entre A et B ?



Théorie des Graphes - 2015/2016

Exercice 2 : dénombrement de chemins

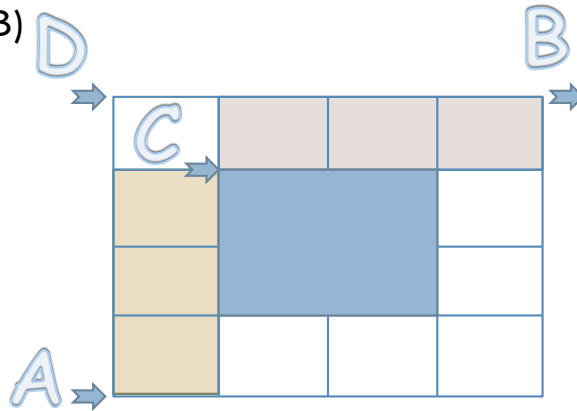
41

□ #AB est le nombre de plus courts chemins entre A et B.

□ $\#AB = 2 * (\#AC * \#CB + \#ADB)$

□ $\#AB = 2 * (C(1, 3+1)^2 + 1)$

□ $\#AB = 2 * (16 + 1) = 34$



Théorie des Graphes - 2015/2016

Graphes eulériens

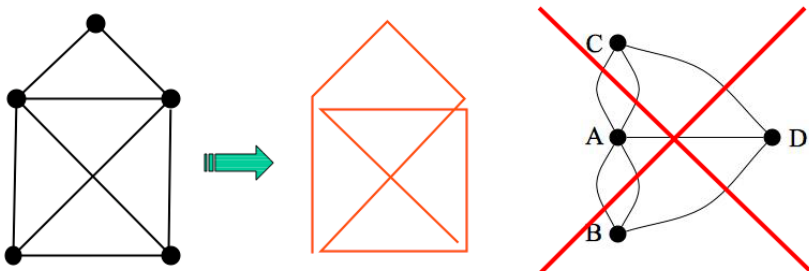
Théorie des Graphes - 2015/2016

Chaîne et cycle eulérien

43

Soit un graphe orienté $G=(X, A)$.

- Une **chaîne eulérienne** est une chaîne empruntant une fois et une fois seulement chaque arête de G .
- Un **cycle eulérien** est une chaîne eulérienne dont les extrémités coïncident.
- Un graphe possédant un cycle eulérien est appelé **graphe eulérien**.



Théorie des Graphes - 2015/2016

Existence chaîne eulérienne

44

Théorème 1

- Un graphe non orienté connexe possède une chaîne eulérienne si et seulement si le nombre de sommets de degré impair est égal à 0 ou 2.
- Il admet un cycle eulérien si et seulement si tous ses sommets ont un degré pair.

Condition nécessaire

En chaque sommet, le nombre d'arcs incidents doit être égal au nombre d'arcs sortants, les sommets doivent donc être de degré pair.

Dans le cas d'une chaîne, les deux extrémités font exception ; on part ou on arrive une fois de plus, d'où un degré impair pour ces deux sommets extrémités.

Théorie des Graphes - 2015/2016

Trouver une chaîne eulérienne

45

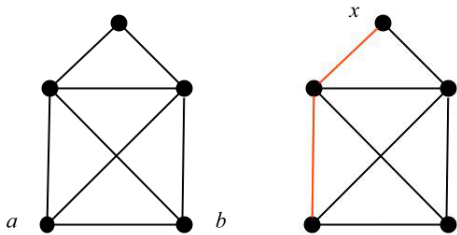
Condition nécessaire

Soit a et b deux sommets de degré impair (s'il n'y en a pas a et b sont confondus).

Soit L la chaîne parcourue en partant de a (avec l'interdiction d'emprunter deux fois la même arête).

Si l'on arrive à un sommet $x \neq b$, on a utilisé un nombre impair d'arêtes incidentes à x . On peut donc « repartir » par une arête non déjà utilisée.

Quand on ne peut plus bouger, c'est qu'on est en b . Si toutes les arêtes ont été utilisées, on a parcouru une chaîne eulérienne.

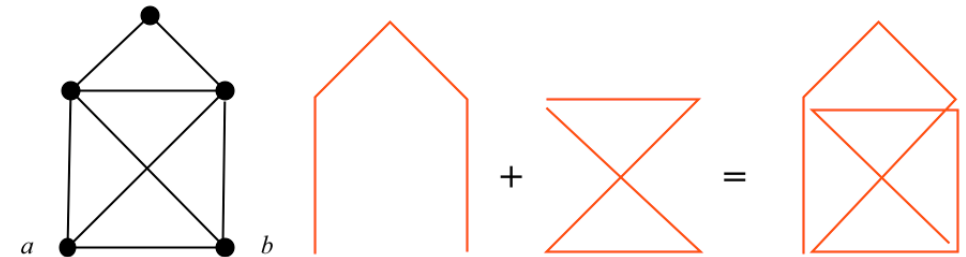


Théorie des Graphes - 2015/2016

Trouver une chaîne eulérienne

46

Si toutes les arêtes n'ont pas été utilisées, on greffe, dans la chaîne, des cycles eulériens

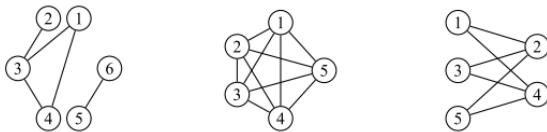


Théorie des Graphes - 2015/2016

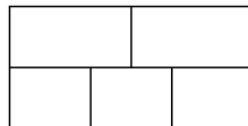
Exercices

47

- Les graphes suivants sont-ils eulériens ?



- Est-il possible de tracer une courbe, sans lever le crayon, qui coupe chacun des 16 segments de la figure suivante exactement une fois ?



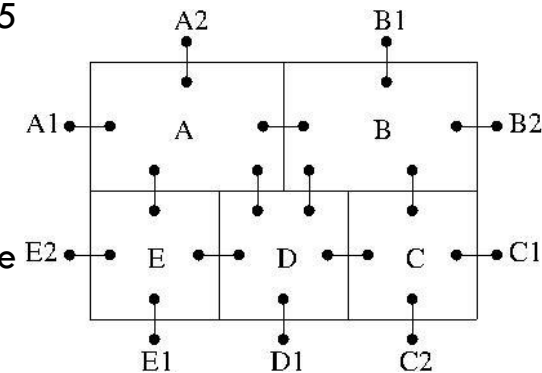
Théorie des Graphes - 2015/2016

Exercices

48

- Le degré de A, B et D est 5
- Le degré de E et C est 4
- Le degré de A1, A2, B1, B2, C1, C2, D1, E1 et E2 est 9.

Il est impossible de tracer une courbe, sans lever le crayon, qui coupe les 16 segments car il y a plus de deux sommets de degré impair.



Théorie des Graphes - 2015/2016

Exercices

49

- Soit G un graphe non eulérien. Est-il toujours possible de rendre G eulérien en lui rajoutant un sommet et quelques arêtes ?
- On considère des dominos dont les faces sont numérotées 1, 2, 3, 4 ou 5.
 - ▣ En excluant les dominos doubles, de combien de dominos dispose-t-on ?
 - ▣ Montrez que l'on peut arranger ces dominos de façon à former une boucle fermée (en utilisant la règle habituelle de contact entre les dominos).
 - ▣ Pourquoi n'est-il pas nécessaire de considérer les dominos doubles ?
 - ▣ Si l'on prend maintenant des dominos dont les faces sont numérotées de 1 à n , est-il possible de les arranger de façon à former une boucle fermée ?

Théorie des Graphes - 2015/2016

Correction : rendre un graphe eulérien

50

- Soit G un graphe non eulérien. Est-il toujours possible de rendre G eulérien en lui rajoutant un sommet et quelques arêtes ?
 - ▣ On ajoute un nouveau sommet que l'on relie à tous les sommets de degré impair. Ces sommets ont maintenant un degré pair dans le graphe modifié.
 - ▣ Dans le graphe original, il y avait un nombre pair de sommets de degré impair, le nouveau sommet a donc un degré pair.
 - ▣ Ainsi, tous les sommets ont maintenant un degré pair et le graphe est eulérien.

Théorie des Graphes - 2015/2016

Correction : dominos

51

- Si l'on prend maintenant des dominos dont les faces sont numérotées de 1 à n , est-il possible de les arranger de façon à former une boucle fermée ?
 - ▣ Les sommets du graphes sont les nombres de 1 à n .
 - ▣ Chaque sommet est relié à tous les autres sommets. Chaque arc (i,j) représente un domino.
 - ▣ En fait, le graphe est complet. Le degré d'un sommet est $n-1$.
 - ▣ Si n est paire, tous les sommets ont un degré impair et le graphe n'est pas eulérien.
 - ▣ Si n est impaire, tous les sommets ont un degré pair et le graphe est eulérien.

Théorie des Graphes - 2015/2016

Chemin et circuit eulérien

52

- Soit un graphe orienté $G=(X, A)$.
- Un chemin dans un graphe orienté est dit **eulérien** s'il passe exactement une fois par chaque arc.
- Un graphe orienté est dit **eulérien** s'il admet un **circuit eulérien**.

Théorie des Graphes - 2015/2016

Chemin et circuit eulérien

53

- Un graphe orienté connexe admet un **chemin eulérien** (mais pas de circuit eulérien) si, et seulement si, pour tout sommet sauf deux (a et b), le degré entrant est égal au degré sortant et

$$d_e(a)=d_s(a)-1 \quad \text{et} \quad d_e(b)=d_s(b)+1$$

- Un graphe orienté connexe admet un **circuit eulérien** si, et seulement si, pour tout sommet, le degré entrant est égal au degré sortant.

Théorie des Graphes - 2015/2016

Chemin hamiltonien

Théorie des Graphes - 2015/2016

Chemin hamiltonien

55

Soit $G=(X, A)$ un graphe connexe d'ordre n .

- On appelle **chemin hamiltonien** (**chaîne hamiltonienne**) un chemin (une chaîne) passant une fois, et une fois seulement, par chacun des sommets de G .
- Un chemin hamiltonien (une chaîne hamiltonienne) est donc un chemin (une chaîne) élémentaire de longueur $n-1$.
- Un **circuit hamiltonien** (un **cycle hamiltonien**) est un circuit (un cycle) qui passe une fois, et une seule fois, par chacun des sommets de G .
- On dit qu'un graphe G est hamiltonien s'il contient un cycle hamiltonien (cas non orienté) ou un circuit hamiltonien (cas orienté).

Théorie des Graphes - 2015/2016

Traveling Salesman Problem (TSP)

56

- **Données** : une liste de villes et leurs distances deux à deux.
- **Question** : trouver le plus petit tour qui visite chaque ville exactement une fois.
- **Reformulation plus mathématique** :
 - Etant donné un graphe complet pondéré, trouver un cycle hamiltonien de poids minimum

Théorie des Graphes - 2015/2016

TSP

57



Chaque ville est visitée une et une seule fois
Un seul tour (pas de sous-tour)

Théorie des Graphes - 2015/2016

TSP

58



Chaque ville est visitée une et une seule fois
Un seul tour (pas de sous-tour)

Théorie des Graphes - 2015/2016

TSP

59

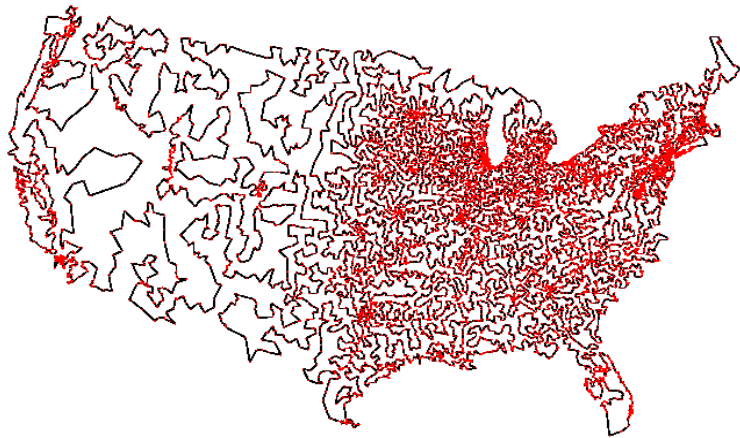
- Certains problèmes sont équivalents au problème du TSP.
Exemple : problème d'ordonnancement : trouver l'ordre dans lequel on doit construire des objets avec des presses hydrauliques
- La version « pure » du TSP n'est pas fréquente en pratique.
On a souvent des problèmes qui sont
 - ▣ Non euclidiens
 - ▣ Asymétriques
 - ▣ Recouvrement de sous-ensembles de nœuds et non pas de tous les nœuds
- Ces variations ne rendent pas le problème plus facile.
- Problème assez commun
 - ▣ Vehicle routing (time windows, pickup and delivery...)

Théorie des Graphes - 2015/2016

60



Procter and Gamble Contest 1962



USA 13,509 cities. Solved in 1998
Théorie des Graphes - 2015/2016

German Tour

- Le 20 Avril 2001, David Applegate, Robert Bixby, Vašek Chvátal, et William Cook ont annoncé la résolution du TSP pour les 15 112 villes d'Allemagne.
- Réseau de 110 processeurs (550 Mhz) à l'université de Rice et de Princeton.
- Le temps total d'utilisation des ordinateurs a été de **22.6 années.**

Théorie des Graphes - 2015/2016



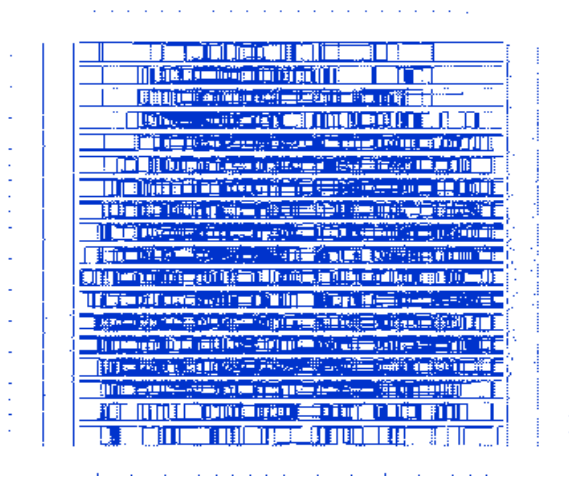
Germany 15,112 cities. Solved in 2001

Théorie des Graphes - 2015/2016



Sweden 24,978 cities. Solved in 2004

Théorie des Graphes - 2015/2016



VLSI 85,900. Solved in 2006
Théorie des Graphes - 2015/2016

TSP

- Il existe plusieurs solveurs
- Le plus connu est Concorde de William Cook (gratuit)

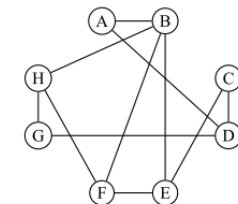
Chemin hamiltonien

Ordonnancement de tâches

- On cherche un ordre dans lequel on peut effectuer n tâches données (deux tâches quelconques ne pouvant être effectuées simultanément) tout en respectant un certain nombre de contraintes d'antériorité.
- Si l'on construit le graphe G dont l'ensemble des sommets correspond à l'ensemble des tâches, et où il existe un arc (i, j) si la tâche i peut être effectuée avant la tâche j , le problème revient à déterminer un chemin hamiltonien de G .

Exercice

- Huit personnes se retrouvent pour un repas de mariage. Le graphe ci-dessous précise les incompatibilités d'humeur entre ces personnes (une arête reliant deux personnes indique qu'elles ne se supportent pas).

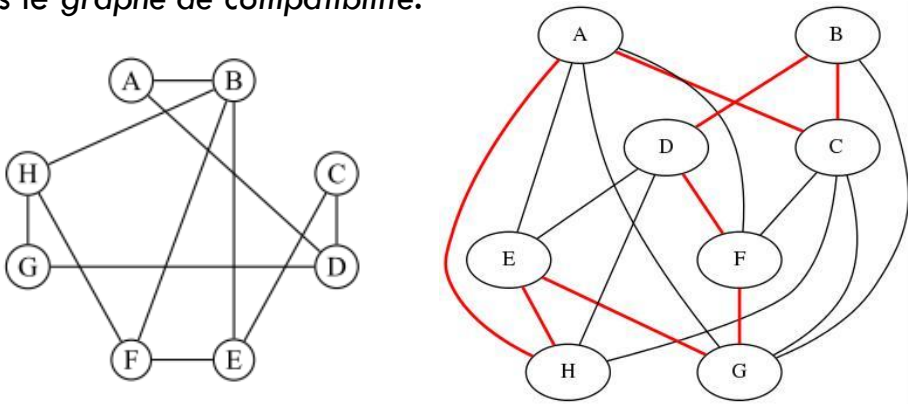


- Proposez un plan de table (la table est ronde) en évitant de placer côte à côte deux personnes incompatibles.

Exercice

69

On cherche un cycle hamiltonien dans le graphe de compatibilité.



Théorie des Graphes - 2015/2016

Connexité

Théorie des Graphes - 2015/2016

Connexité

71

Un graphe G est **connexe** s'il existe au moins une chaîne entre une paire quelconque de sommets de G .

La relation :

$$x_i \text{ R } x_j \Leftrightarrow \begin{cases} \text{soit } x_i = x_j \\ \text{soit il existe une chaîne joignant } x_i \text{ à } x_j \end{cases}$$

est une relation d'équivalence (réflexivité, symétrie, transitivité). Les classes d'équivalence induites sur X par cette relation forment une partition de X en X_1, X_2, \dots, X_p .

Le nombre p de classes d'équivalence distinctes est appelé **nombre de connexité** du graphe.

Un graphe est dit connexe si et seulement si son nombre de connexité est égal à 1.

Les sous-graphes G_i engendrés par les sous-ensembles X_i sont appelés les **composantes connexes** du graphe. Chaque composante connexe est un graphe connexe.

Théorie des Graphes - 2015/2016

Connexité

72

- Un **point d'articulation** d'un graphe est un sommet dont la suppression augmente le nombre de composantes connexes.
- Un **isthme** est une arête dont la suppression a le même effet.
- Un **ensemble d'articulation** d'un graphe connexe G est un ensemble de sommets tel que le sous-graphe G' déduit de G par suppression des sommets de E , ne soit plus connexe.



Théorie des Graphes - 2015/2016

Forte connexité

73

Un graphe orienté est **fortement connexe** s'il existe un chemin joignant deux sommets quelconque.

La relation :

$$x_i R x_j \Leftrightarrow \begin{cases} \text{soit } x_i = x_j \\ \text{soit il existe à la fois un chemin joignant } x_i \text{ à } x_j \text{ et un chemin joignant } x_j \text{ à } x_i \end{cases}$$

est une relation d'équivalence et les classes d'équivalence induites sur X par cette relation forment une partition de X en X_1, X_2, \dots, X_q .

Les sous-graphes engendrés par les sous-ensembles G_1, G_2, \dots, G_q sont appelés les **composantes fortement connexes** du graphe.

Théorie des Graphes - 2015/2016

Graphe Réduit

74

On appelle **graphe réduit** le quotient du graphe G par la relation de forte connexité $G_r = G/R$

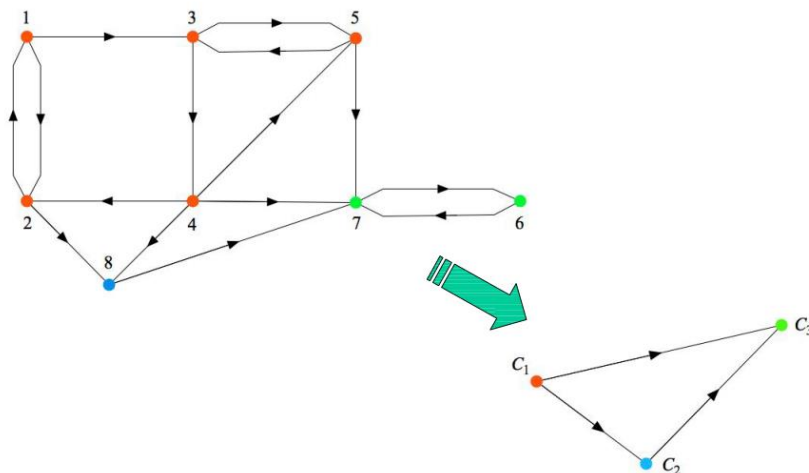
Les sommets de G_r sont donc les composantes fortement connexes et il existe un arc entre deux composantes fortement connexes si et seulement s'il existe au moins un arc entre un sommet de la première composante et un sommet de la seconde.

On vérifie que le graphe G_r est sans circuit.

Théorie des Graphes - 2015/2016

Réduction

75

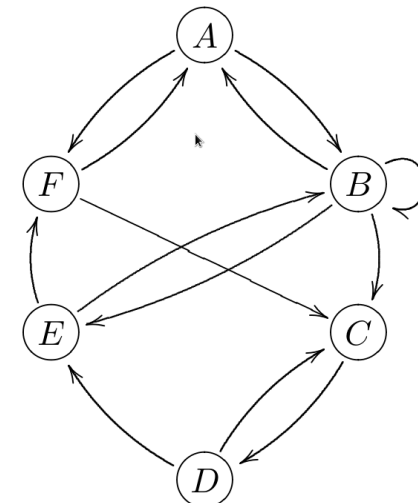


Théorie des Graphes - 2015/2016

Exercices

76

- Donner l'ensemble des successeurs et de prédécesseurs de chaque sommet.
- Calculer les demi-degrés intérieurs et extérieurs de chaque sommet.
- Donner un exemple de chemin simple mais non élémentaire.
- Existe-t-il un circuit hamiltonien dans G ?
- Tracer le graphe non orienté déduit de G .
- G est-il connexe ? Fortement connexe?



Théorie des Graphes - 2015/2016

Arbre et arborescence

Cycle et nombre cyclomatique

78

- On dit que p cycles $\mu_1, \mu_2, \dots, \mu_p$ sont **dépendants** s'il existe, entre leurs vecteurs associés, une relation vectorielle de la forme :
$$\lambda_1 \mu_1 + \lambda_2 \mu_2 + \dots + \lambda_p \mu_p = 0$$
avec les λ_i non tous nuls.
- Si la satisfaction de la relation précédente implique $\lambda_i = 0, i = 1, \dots, p$, les p cycles sont dits **indépendants**.
- Une **base de cycles** est un ensemble minimal de cycles indépendants tel que tout vecteur représentatif d'un cycle puisse s'exprimer comme combinaison linéaire des cycles de la base.
- On appelle **nombre cyclomatique** d'un graphe G , la dimension de la base de cycles.

Nombre cyclomatique

79

- Soit G un graphe avec n sommets, m arêtes (ou arcs) et p composantes connexes
- La dimension de la base de cycle est $v(G) = m - n + p$

Arbre

80

- Un **arbre** est un graphe connexe sans cycles.
- Une **forêt** est un graphe dont chaque composante est un arbre
- **Théorème.** $G=(X,A)$ est un arbre si et seulement si il existe une unique chaîne reliant toute paire de sommet.
- Preuve:
 - Si 2 chaînes alors on aurait un cycle
 - connexe équivaut à il existe toujours une chaîne entre toute paire de sommet

Arbre

81

- **Théorème** : Soit $H=(X,U)$ un graphe d'ordre ≥ 2 ; les propriétés suivantes sont équivalentes pour caractériser un arbre
 - ▣ H est connexe et sans cycle
 - ▣ H est sans cycle et admet $n-1$ arcs
 - ▣ H est connexe et admet $n-1$ arcs
 - ▣ H est connexe et en ajoutant un arc on crée un cycle (et un seul)
 - ▣ H est sans cycle et si on supprime un arc quelconque, il n'est plus connexe
 - ▣ Tout couple de sommets est relié par une chaîne et une seule
- Preuve : voir Berge

Théorie des Graphes - 2015/2016

Graphe fortement connexe

82

- **Théorème** : Pour un graphe G avec au moins un arc, les conditions suivantes sont équivalentes :
 - ▣ G est fortement connexe
 - ▣ Par tout arc passe un circuit
- Preuve : voir Berge

Théorie des Graphes - 2015/2016

Arborescence

83

- Dans un graphe $G=(X,U)$, on appelle racine, un point α tel que tout autre sommet du graphe puisse être atteint par un chemin issu de α . Une racine n'existe pas toujours.
- Une **arborescence** est un arbre muni d'une racine
- Un graphe G est dit **quasi-fortement connexe** si pour tout couple de sommet x et y , il existe un sommet z d'où partent à la fois un chemin allant en x et un chemin allant en y

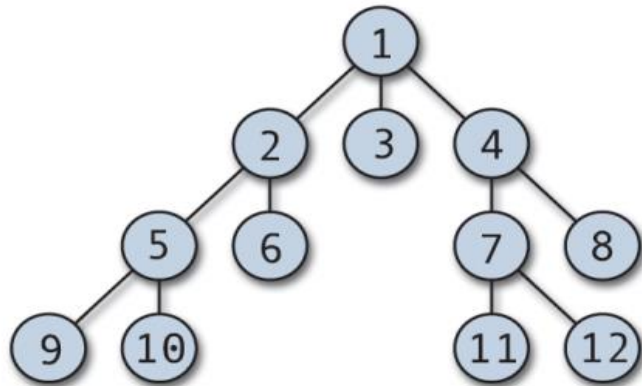
Théorie des Graphes - 2015/2016

Cheminement

Théorie des Graphes - 2015/2016

Arbre

85



Théorie des Graphes - 2015/2016

Arbre

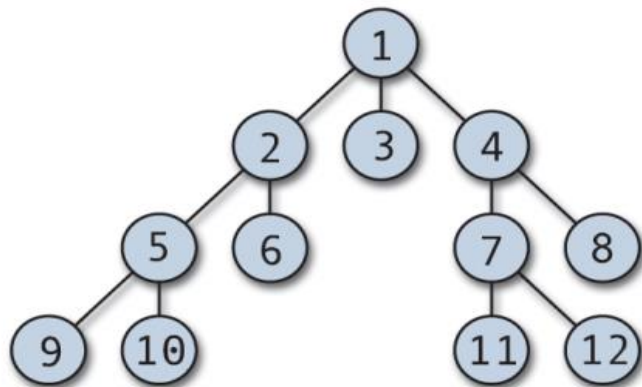
86

- La **racine** r de l'arbre est l'unique nœud ne possédant pas de parent
- Tout sommet x qui n'est pas la racine a
 - un unique parent, noté $\text{parent}(x)$ (appelé père parfois)
 - 0 ou plusieurs fils. $\text{fils}(x)$ désigne l'ensemble des fils de x
- Si x et y sont des sommets tels que x soit sur le chemin de r à y alors
 - x est un ancêtre de y
 - y est un descendant de x
- Un sommet qui n'a pas de fils est une feuille

Théorie des Graphes - 2015/2016

Arbre

87



1 est la racine
9,10,6,3,11,12,8 sont les feuilles
11 est un descendant de 4 mais pas de 2
2 est un ancêtre de 10

Théorie des Graphes - 2015/2016

Arbre

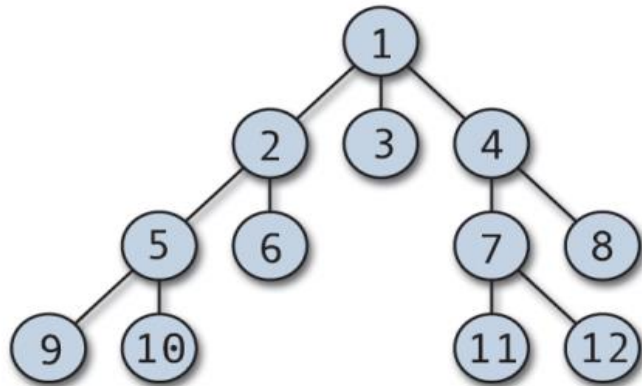
88

- Quand il n'y a pas d'ambiguïté, on regarde les arêtes d'un arbre comme étant orienté de la racine vers les feuilles
- La **profondeur** d'un sommet (depth) est définie récursivement par
 - $\text{prof}(v) = 0$ si v est la racine
 - $\text{prof}(v) = \text{prof}(\text{parent}(v)) + 1$

Théorie des Graphes - 2015/2016

Arbre

89



1 est la racine
2,3,4 sont à la profondeur 1
5,6,7,8 à la profondeur 2
La hauteur de 2 est 2, celle de 9 est 4, celle de 3 est 2, celle de 1 est 1.

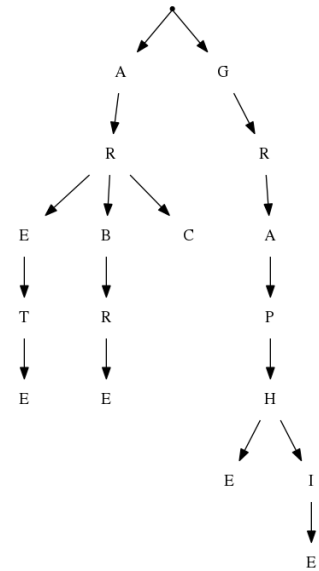
Théorie des Graphes - 2015/2016

Exemple d'arbre : dictionnaire

90

L'arbre ci-contre représente un dictionnaire contenant les 5 mots suivants: arbre, arc, arete, graphe, graphie.

- Ajouter le mot « arborer » au dictionnaire. Proposez un algorithme pour l'insertion d'un mot.
- Ajouter le mot « are » dans le dictionnaire. Que constatez-vous ? Proposez une solution ?
- Proposez un algorithme déterminant si un mot appartient au dictionnaire.
- Quel est le nombre de sommets maximum d'un dictionnaire de profondeur p ?



Théorie des Graphes - 2015/2016

Arbre : parcours

91

- Parcours (tree traversal) : on traverse l'ensemble des sommets de l'arbre
- Parcours en largeur d'abord
- Parcours en profondeur d'abord
 - ▣ Préfixe
 - ▣ Infixe (arbre binaire seulement)
 - ▣ Postfixe

Théorie des Graphes - 2015/2016

Arbre : parcours en largeur d'abord

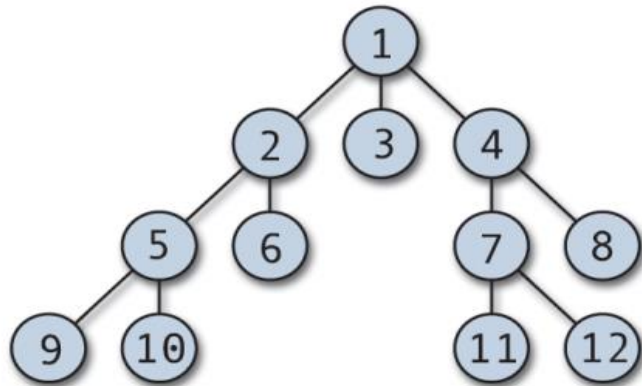
92

- Largeur d'abord (bfs = breadth-first search)
- On visite la racine, puis on répète le processus suivant jusqu'à avoir visité tous les sommets :
visiter un fils non visité du sommet le moins récemment visité qui a au moins un fils non visité
- On visite tous les sommets à la profondeur 1, puis tous ceux à la profondeur 2, puis tous ceux à la profondeur 3 etc...

Théorie des Graphes - 2015/2016

Arbre

93



Largeur d'abord: ordre de visite
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12

Théorie des Graphes - 2015/2016

Arbre : parcours en largeur d'abord

94

```
□ Bfs(T) : array  
  r ← racine(T)  
  créer un file F et ajouter r dans F  
  i ← 0  
  tant que (F n'est pas vide)  
    x ← premier(F); supprimer x de F  
    array[i] ← x  
    i++  
    pour chaque fils y de x  
      ajouter y dans F  
  fin pour  
fin tant que
```

Théorie des Graphes - 2015/2016

Arbre : largeur d'abord avec passes

95

```
□ Bfs(T) : array  
  r ← racine(T)  
  créer deux files F1 et F2 et ajouter r dans F1  
  i ← 0  
  faire  
    tant que (F1 n'est pas vide)  
      x ← premier(F1); supprimer x de F1  
      array[i] ← x  
      i++  
      pour chaque fils y de x  
        ajouter y dans F2  
      fin pour  
    fin tant que  
    F1 ← F2 // fin d'une passe début de la nouvelle : la  
    F2 devient vide // profondeur change  
  tant que F1 n'est pas vide
```

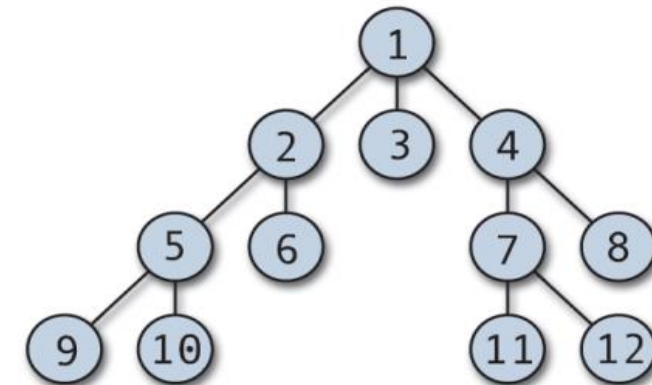
Théorie des Graphes - 2015/2016

Arbre : parcours en largeur d'abord

96

Largeur d'abord: ordre de visite

- Passe 1 : 1
- Passe 2 : 2,3,4
- Passe 3 : 5,6,7,8
- passe 4 : 9,10,11,12



Théorie des Graphes - 2015/2016

Arbre : parcours en profondeur d'abord

97

- Profondeur d'abord (dfs = depth-first search)
- Défini de façon récursive
 - `visit(sommet x)`
 - `previsit(x)`
 - pour chaque fils `y` de `x`
 - `visit(y)`
 - `postvisit(x)`
- Premier appel : `visit(racine(T))`

Théorie des Graphes - 2015/2016

Arbre : parcours en profondeur d'abord

98

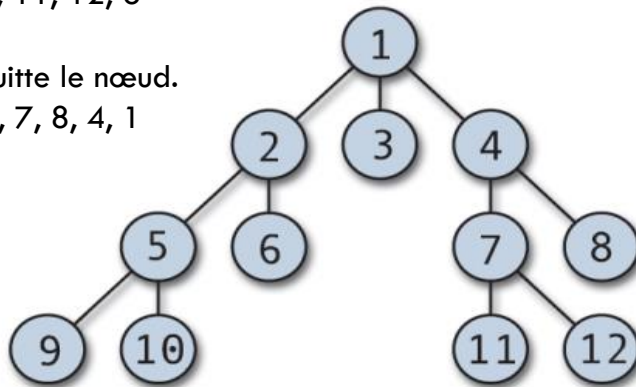
- Ordre préfixé ou postfixé dépend des fonction `previsit` et `postvisit`
- Si `previsit(x)` : met `x` dans le array et incremente `i` alors array contient l'ordre préfixé
- Si c'est `postvisit` qui le fait alors array contiendra l'ordre postfixé

Théorie des Graphes - 2015/2016

Arbre: parcours en profondeur d'abord

99

- ordre de visite préfixé
 - On marque quand on atteint le nœud.
 - 1, 2, 5, 9, 10, 6, 3, 4, 7, 11, 12, 8
- ordre de visite postfixé
 - On marque quand on quitte le nœud.
 - 9, 10, 5, 6, 2, 3, 11, 12, 7, 8, 4, 1



Théorie des Graphes - 2015/2016

Arbre : parcours en profondeur d'abord

100

- `Dfs(T)` : array
 - `r ← racine(T)`
 - créer une pile `P` et ajouter `r` dans `P`
 - `i ← 0`
 - tant que (`P` n'est pas vide)
 - `x ← premier(P)`; supprimer `x` de `P`
 - `array[i] ← x`
 - `i++`
 - pour chaque fils `y` de `x`
 - ajouter `y` dans `P`
 - fin pour
 - fin tant que

Théorie des Graphes - 2015/2016

Arbre : implémentation

101

Représentation des fils

Par une liste :

- Le parent possède un premier fils
- Chaque sommet possède un pointeur vers son frère suivant (liste chaînée des fils)

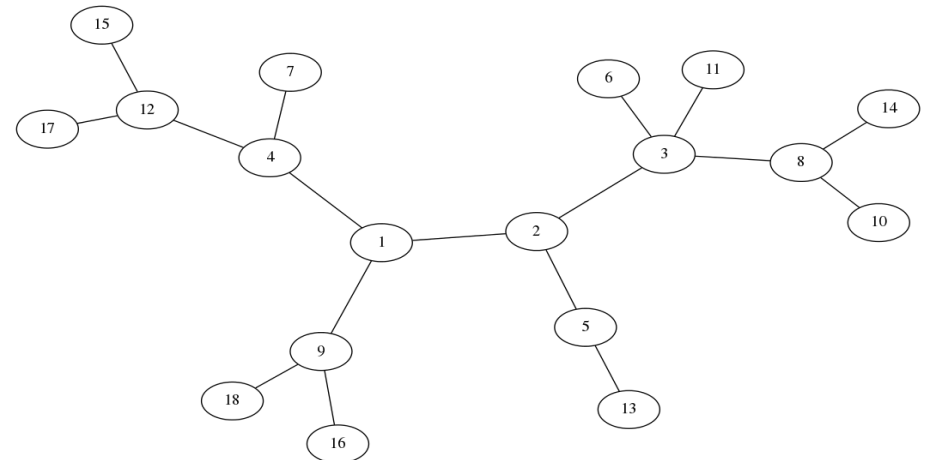
Par un tableau si le nombre de fils est connu à l'avance (arbre k-aire)

Dans le cas binaire, le parent possède le fils gauche et le fils droit

Théorie des Graphes - 2015/2016

Exercice : parcours d'arbre

102



- Effectuez un parcours en largeur/profondeur d'abord en partant de 1.
- Vous devez visiter les successeurs d'un nœud dans leur ordre naturel.

Théorie des Graphes - 2015/2016

Cheminement dans un graphe

103

- Les algorithmes que l'on a vu pour les arbres (DFS et BFS) peuvent s'appliquer aux graphes

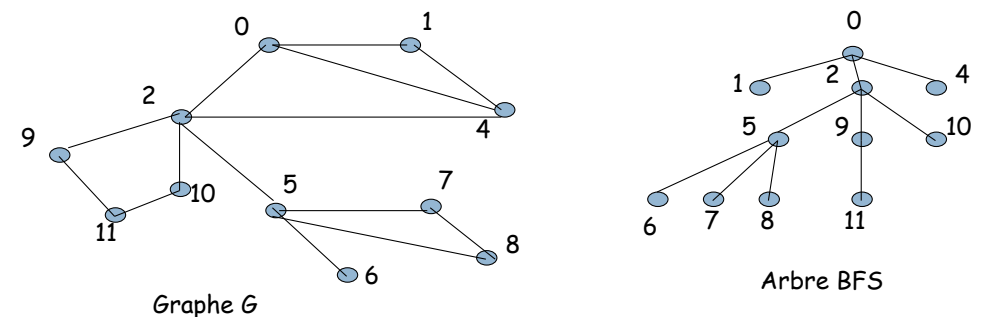
- Il faut faire un peu attention

On ne doit pas visiter deux fois le même sommet

Théorie des Graphes - 2015/2016

Graphe: largeur d'abord

104



Théorie des Graphes - 2015/2016

Graphe : largeur d'abord

105

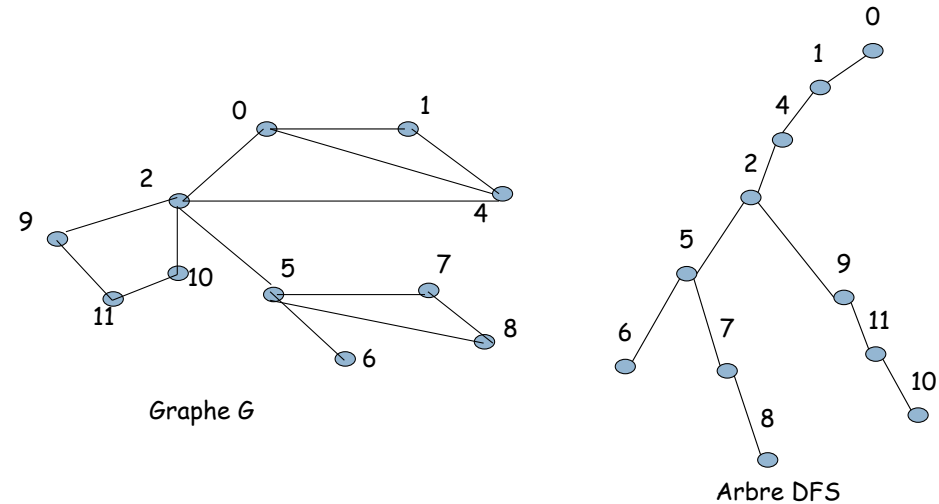
```

□ Bfs(G,s) : array
  r ← s
  pour tous les sommets x marque[x] ← faux
  créer un file F; enfiler(F,r); marque[r] ← vrai
  i ← 0
  tant que (F n'est pas vide)
    x ← premier(F); defiler(F)
    array[i] ← x
    i++
    pour chaque voisin y de x
      si marque[y] est faux
      alors marque[y] ← vrai
      enfiler(F,y)
  fin pour
  fin tant que
    
```

Théorie des Graphes - 2015/2016

Graphe : profondeur d'abord

106



Théorie des Graphes - 2015/2016

Graphe : profondeur d'abord

107

```

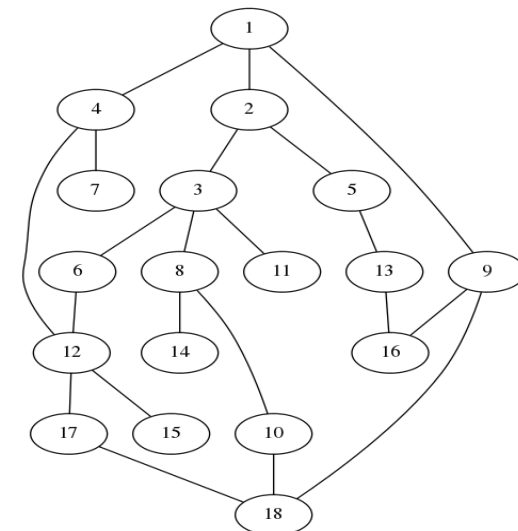
□ Dfs(G,s) : array
  r ← s
  pour tous les sommets x marque[x] ← faux
  créer un pile P; push(P,r); marque[r] ← vrai
  i ← 0
  tant que (P n'est pas vide)
    x ← top(P); pop(P);
    array[i] ← x
    i++
    pour chaque voisin y de x
      si marque[y] est faux
      alors marque[y] ← vrai
      push(P,y)
  fin pour
  fin tant que
    
```

Théorie des Graphes - 2015/2016

Exercice : parcours de graphe

108

- Effectuez un parcours en largeur/profondeur d'abord en partant de 1.
- Précisez :
 - l'arbre du parcours
 - la liste des prédécesseurs
 - l'ordre préfixe/postfixe
- Visitez les successeurs d'un nœud dans l'ordre :
 - naturel (croissant).
 - ante-naturel (décroissant)

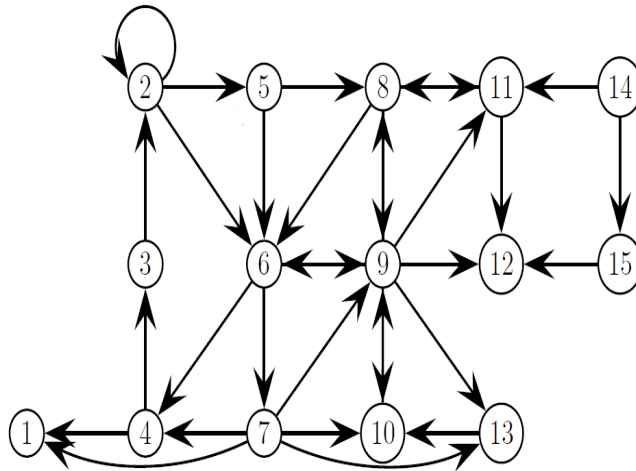


Théorie des Graphes - 2015/2016

Exercice : parcours de graphe orienté

109

- Effectuez un parcours en largeur/profondeur d'abord en partant de 5.
- Précisez :
 - l'arbre du parcours
 - la liste des prédécesseurs
 - l'ordre préfixe/postfixe
- Visitez les successeurs d'un nœud dans l'ordre :
 - naturel (croissant).
 - ante-naturel (décroissant)



Théorie des Graphes - 2015/2016

Recherche de connexité

Théorie des Graphes - 2015/2016

Graphe connexe

111

- Ecrire un algorithme qui détermine si un graphe est connexe
- Ecrire un algorithme qui calculent les composantes connexes d'un graphe

Théorie des Graphes - 2015/2016

Composantes fortement connexes

112

- Ecrire un algorithme qui teste si un graphe est fortement connexe
- En écrire un meilleur !

Théorie des Graphes - 2015/2016

Composantes fortement connexes

113

- On calcule G' le graphe inverse (transpose graph)
- On marque « non marqué » tous les sommets
- Début: On prend un sommet s non marqué
- On calcule tous les sommets que l'on peut atteindre dans G et dans G' à partir de s
- On marque « marqué » tous les sommets atteints deux fois ainsi que s
- On retourne en Début tant qu'il reste des sommets « non marqués »

- Complexité ? En théorie ? En pratique ?

Théorie des Graphes - 2015/2016

Composantes fortement connexes

114

- Algorithme de R.E **Tarjan**
 - ▣ Premier algorithme linéaire : complexité en $O(n+m)$
 - ▣ [ICS 161: Design and Analysis of Algorithms](#), D. Eppstein.
- Algorithme de **Kosaraju**
 - ▣ [Kosaraju's algorithm](#), Wikipedia.
- Algorithme de **Gabow** (Path-based strong component algorithm)

Théorie des Graphes - 2015/2016

DFS types d'arcs

115

- On visite v , on atteint w
 - ▣ w n'a jamais été atteint
 - ▣ w est un ancêtre de v dans la DFS
 - ▣ w est déjà été totalement visité

- Idée : on marque les nœuds avec un indice que l'on augmente. On regarde quel est l'indice minimal que l'on peut atteindre à partir d'un nœud.

Théorie des Graphes - 2015/2016

Algorithme de Tarjan

116

- $num := 0$; $P :=$ pile vide; partition := ensemble vide
- pour chaque sommet v de G
 - si $v.num$ n'est pas défini
 - parcours(G, v)
 - renvoyer partition
- fin de fonction

Théorie des Graphes - 2015/2016

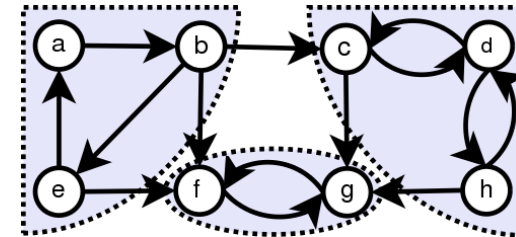
Function parcours(sommet v)

```

□ v.num := num
  v.numAccessible := num
  num := num + 1
  P.push(v), v.dans_P := oui
  // Parcours récursif
  pour chaque w successeur de v
    si w.num n'est pas défini
      parcours(w)
      v.numAccessible := min(v.numAccessible, w.numAccessible)
    sinon si w.dans_P = oui
      v.numAccessible := min(v.numAccessible, w.num)
  si v.numAccessible = v.num
    // C'est une racine, calcule la composante fortement connexe associée
    C := ensemble vide
    répéter
      w := P.pop(), w.dans_P := non
      ajouter w à C
    tant que w différent de v
    ajouter C à partition
  
```

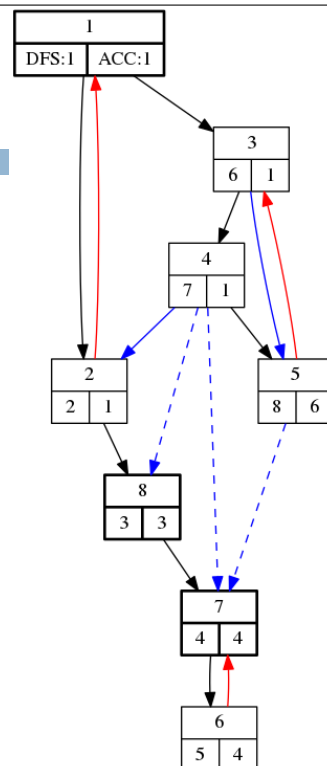
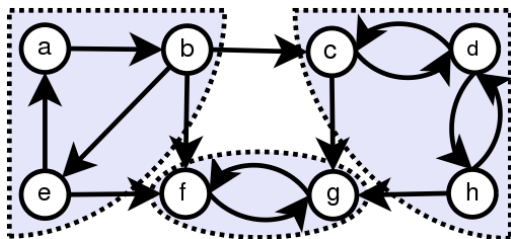
Théorie des Graphes - 2015/2016

Composantes fortement connexes



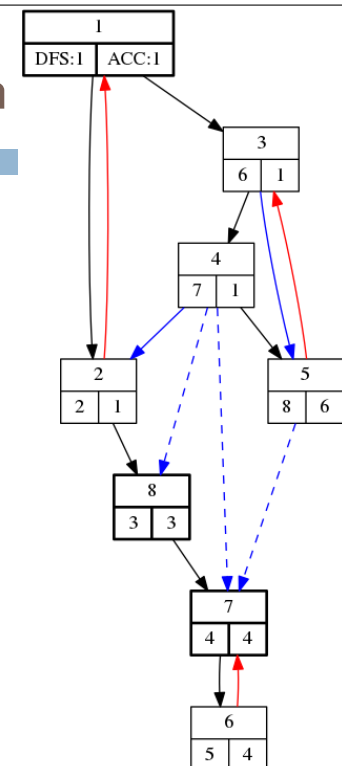
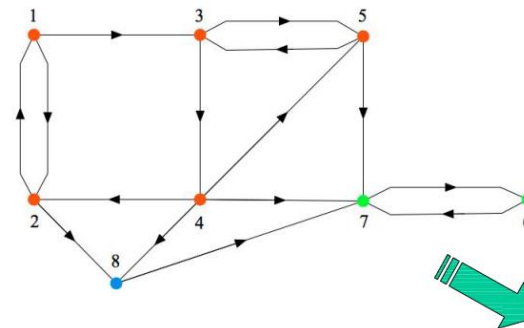
Théorie des Graphes - 2015/2016

CFC : algorithme de Tarjan



Théorie des Graphes - 2015/2016

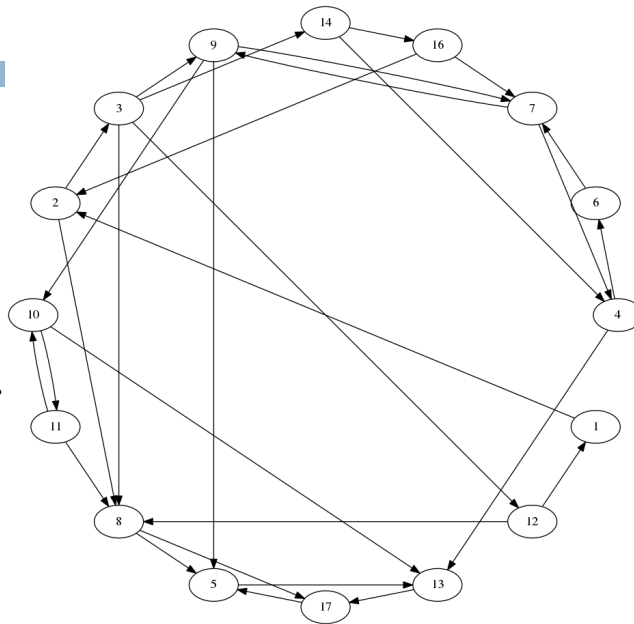
CFC : algorithme de Tarjan



Théorie des Graphes - 2015/2016

Exercice : composantes fortement connexes.

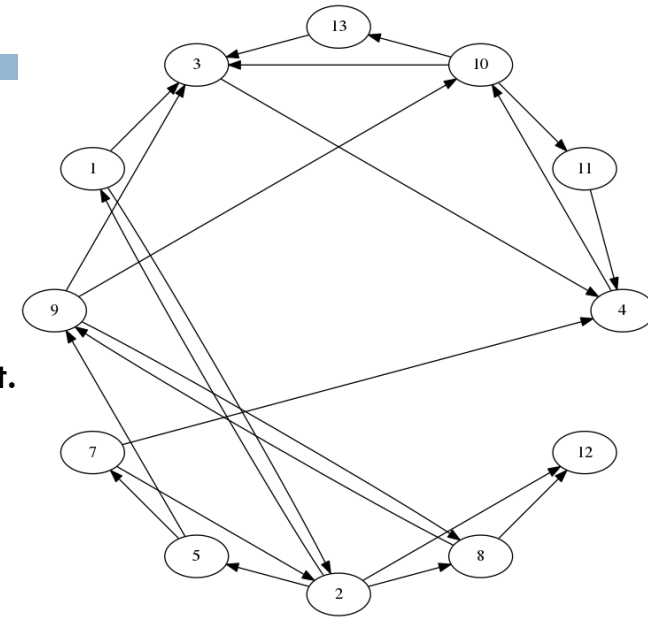
- Trouvez les CFC avec l'algorithme de Tarjan.
- Dessinez l'arbre DFS.
- Dessinez le graphe réduit.



Théorie des Graphes - 2015/2016

Exercice : encore des CFC.

- Trouvez les CFC avec l'algorithme de Tarjan.
- Dessinez l'arbre DFS.
- Dessinez le graphe réduit.



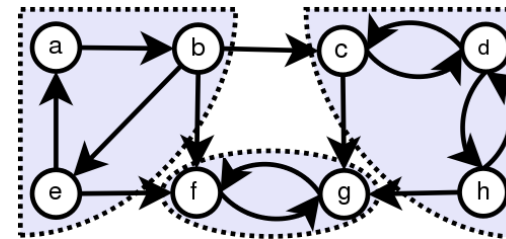
Théorie des Graphes - 2015/2016

Algorithme de Kosaraju

- S : pile vide
- Tant que S ne contient pas tous les sommets
 - Choisir un sommet arbitraire v qui n'est pas dans S
 - Lancer une DFS depuis v. Mettre tous les sommets visités (**à la fin de leur visite**) u dans S
- Inversé la direction de tous les arcs pour obtenir le graphe transposé.
- Tant que S est non vide:
 - Affecter v avec le sommet de S. Dépiler S
 - Lancer une DFS depuis v. (On peut aussi utiliser une BFS)
 - L'ensemble des sommets visités depuis v appartiennent à la cfc contenant v; supprimer ces sommets de G et de S.

Théorie des Graphes - 2015/2016

Algorithme de Kosaraju



On commence à a
a,b,c,d,h,g,f
On ferme f : $S=\{f\}$
On ferme g,h,d,c
 $S=\{f,g,h,d,c\}$
On ouvre e
On ferme e,b,a
 $S=\{f,g,h,d,c,e,b,a\}$

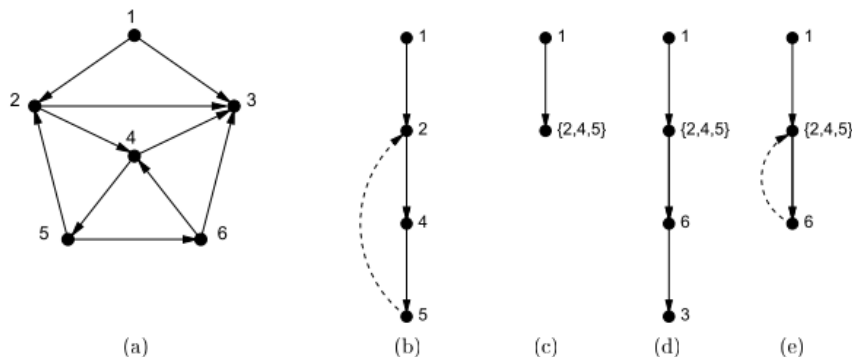
On prend le transposé.
On dépile S: a. On atteint b et e
 $CFC=\{a,b,e\}$; $S=\{f,g,h,d,c\}$
On dépile S: c. On atteint d,h
 $CFC=\{c,d,h\}$; $S=\{f,g\}$
On dépile S: g. On atteint f
 $CFC=\{f,g\}$

Théorie des Graphes - 2015/2016

Path-based strong component algorithm

125

Idée



Théorie des Graphes - 2015/2016

Path-based strong component algorithm

126

procedure STRONG(G)

1. empty stacks S and B ;
2. for $v \in V$ do $I[v] = 0$;
3. $c = n$;
4. for $v \in V$ do if $I[v] = 0$ then DFS(v);

procedure DFS(v)

1. PUSH(v, S); $I[v] = \text{TOP}(S)$; PUSH($I[v], B$); /* add v to the end of P */
2. for edges $(v, w) \in E$ do
3. if $I[w] = 0$ then DFS(w)
4. else /* contract if necessary */ while $I[w] < B[\text{TOP}(B)]$ do POP(B);
5. if $I[v] = B[\text{TOP}(B)]$ then { /* number vertices of the next strong component */
6. POP(B); increase c by 1;
7. while $I[v] \leq \text{TOP}(S)$ do $I[\text{POP}(S)] = c$ };

Théorie des Graphes - 2015/2016

Path-based strong component algorithm

127

- L'algorithme applique une DFS et maintient 2 piles S et P .
- La pile S contient tous les sommets qui n'ont pas encore été affectés à une CFC dans l'ordre dans lequel ils ont été atteints par la DFS.
- La pile P contient tous les sommets dont on ne sait pas s'ils appartiennent à une CFC différentes de celles trouvées
- On met $\text{num}(x)=0$ pour tous les sommets et $C=1$

Théorie des Graphes - 2015/2016

Visit(v)

128

- $\text{num}(v) \leftarrow C$; incrémenter C
- Empiler v sur S ; Empiler v sur P .
- Pour chaque voisin w de v
 - si $\text{num}(w)=0$ alors visit(w)
 - sinon si w n'a pas été placé dans une CFC alors
 - Tant que $\text{num}(\text{sommet}(P)) > \text{num}(w)$
 - dépiler P
- Si v est le sommet de P :
 - Dépiler S jusqu'à dépiler v . Mettre tous les sommets dépilés dans une CFC
 - Dépiler v de P

Théorie des Graphes - 2015/2016

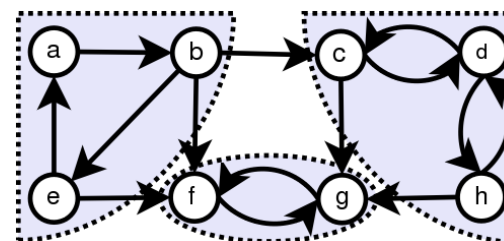
Path-based strong component algorithm

129

- On fait une boucle sur les sommets non encore ouverts par la DFS

Path-based strong component algorithm

130



On commence à a
 $S=\{a\}$; $P=\{a\}$; $\text{num}(a)=1$
 $S=\{a,b\}$; $P=\{a,b\}$; $\text{num}(b)=2$
 $S=\{a,b,c\}$; $P=\{a,b,c\}$; $\text{num}(c)=3$
 $S=\{a,b,c,d\}$; $P=\{a,b,c,d\}$; $\text{num}(d)=4$
 On touche c. d est dépilé de P
 $S=\{a,b,c,d,h\}$; $P=\{a,b,c,h\}$; $\text{num}(h)=5$
 On touche d. h est dépilé de P
 $S=\{a,b,c,d,h,g\}$; $P=\{a,b,c,g\}$; $\text{num}(g)=6$
 $S=\{a,b,c,d,h,g,f\}$; $P=\{a,b,c,g,f\}$; $\text{num}(f)=7$
 On touche g. f est dépilé

$S=\{a,b,c,d,h,g,f\}$; $P=\{a,b,c,g\}$; On quitte g
 g est le sommet de P. On dépile S jusqu'à g $\text{CFC}=\{f,g\}$. $P=\{a,b,c\}$
 On quitte h,d,c. c est le sommet de P. On dépile S jusqu'à c
 $\text{CFC}=\{c,d,h\}$; $P=\{a,b\}$; $S=\{a,b\}$
 on ajoute e. $S=\{a,b,e\}$; $P=\{a,b,e\}$. On touche a. e et b sont dépilés de P
 $P=\{a\}$. On quitte e,b,a. a est le sommet de P
 $\text{CFC}=\{a,b,c\}$

K-connexité

131

- La k-connexité revient à se demander combien il faut éliminer de sommet au minimum pour casser la connexité du graphe
 - 2 : alors graphe 2-connexe
- On peut aussi se poser la question pour les arêtes
 - K-arête connexes : il faut éliminer k arêtes
- Il existe des algorithmes dédiés souvent semblables aux algorithmes présentés avant (2 connexes et algo de Tarjan pour les composantes fortement connexes)

Problème 2-SAT

Un graphe pour résoudre 2-SAT, Philippe Gambette.

Exercices : problème 2-SAT

133

Trouvez les valuations des formules 2-SAT suivantes.

- $(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3)$
- $(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_3)$
- $12 + 1-3 + -14 + 23 + 27 + 2-6 + 34 + 45 + 46 + 4-7 + 6-7$

Théorie des Graphes - 2015/2016

Problèmes de cheminement

Théorie des Graphes - 2015/2016

Problèmes

135

- Connexité et sommets atteignables
- Plus court chemin entre deux sommets
 - Algorithme de Dijkstra
 - Algorithme de Bellman-Ford
 - Combinaison des deux
- Détection de cycles négatifs
- Plus courts chemins entre tous les sommets
- Cas des graphes sans circuit

Théorie des Graphes - 2015/2016

Sommets atteignables

136

- Pour savoir quels sont les sommets atteignables à partir d'un sommet, il suffit d'utiliser une procédure de parcours du graphe (DFS ou BFS)
- Trouver le chemin le plus court est un peu plus complexe
- On cherche le plus court chemin entre deux sommets s et t (source=source; t =sink)

Théorie des Graphes - 2015/2016

Plus court chemin entre deux points

137

- On introduit maintenant une longueur sur les arcs
- Le coût sur les arcs peut être appelé différemment
 - ▣ Distance
 - ▣ Poids
 - ▣ Coût
 - ▣ ...
- important : **la longueur d'un chemin est égal à la somme des longueurs des arcs qui le compose**

Théorie des Graphes - 2015/2016

Plus court chemin entre deux points

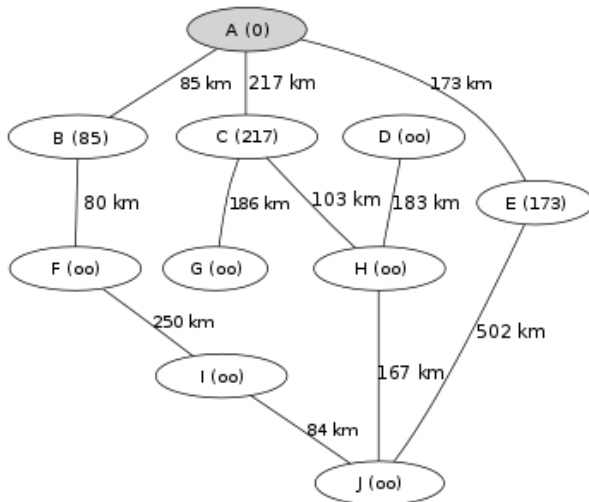
138

- Pour commencer, on supposera que tous les coûts sont positifs. Ensuite on relâchera cette condition.
- Ce n'est pas une obligation : on peut très bien calculer le plus court chemin avec des coûts négatifs.

Théorie des Graphes - 2015/2016

Chemin de A vers J

139



Théorie des Graphes - 2015/2016

Plus court chemin

140

- On ne peut pas énumérer tous les chemins
- On se place dans le cas où toutes les longueurs sont positives
- Un algorithme glouton existe !

Théorie des Graphes - 2015/2016

Problèmes

- Connexité et sommets atteignables
- Plus court chemin entre deux sommets
 - ▣ **Algorithme de Dijkstra**
 - ▣ Algorithme de Bellman-Ford
 - ▣ Combinaison des deux
- Détection de cycles négatifs
- Plus courts chemins entre tous les sommets
- Cas des graphes sans circuit

Théorie des Graphes - 2015/2016

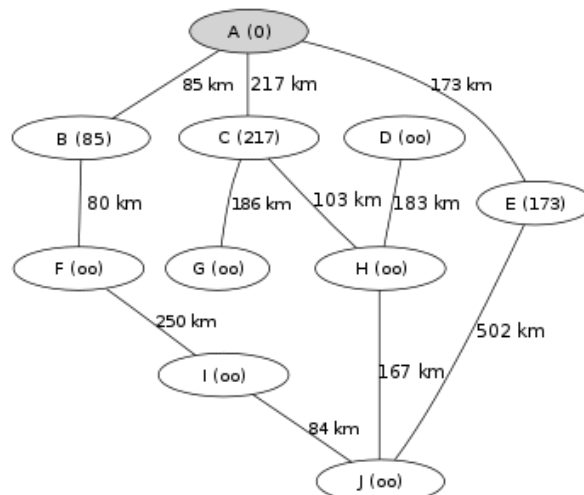
Algorithme de Dijkstra

- On maintien en permanence la distance des sommets avec l'origine
- On a trois ensembles de sommets
 - ▣ Les **ouverts** (ce sont les candidats pour la prochaine étape)
 - On peut les atteindre en passant par des sommets précédemment choisis
 - ▣ Les **fermés** : ce sont des sommets qui ont été choisis
 - ▣ Les **indéfinis** : pour l'instant on ne peut pas les atteindre
- A chaque étape on choisit le sommet ouvert situé à la plus petite distance
- On regarde les sommets qui sont reliés à ce sommet
 - ▣ Ceux qui sont indéfinis deviennent ouvert et on calcule leur distance
 - ▣ Ceux qui sont ouverts ont leur distance modifiée
- On ferme le sommet

Théorie des Graphes - 2015/2016

Chemin de A vers J

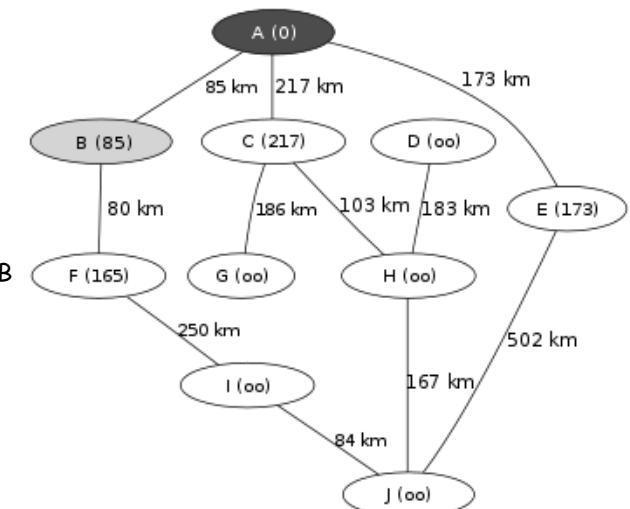
A est ouvert
Sommets reliés à A : B, C et E
Ils sont ouverts avec
 $D(A, B) \leq 85$
 $D(A, C) \leq 217$
 $D(A, E) \leq 173$
Ouvert = {B, E, C}



Théorie des Graphes - 2015/2016

Chemin de A vers J

On ajoute le sommet ouvert à la plus petite distance
Ici c'est B. On ferme B
On s'intéresse aux voisins de B
Il y a F : on ouvre F
 $D(A, F) \leq d(A, B) + d(B, F) \leq 165$
Ouvert = {F, E, C}

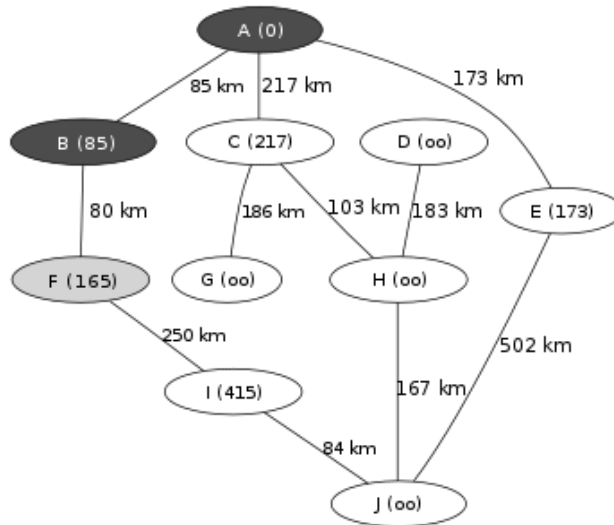


Théorie des Graphes - 2015/2016

Chemin de A vers J

145

On choisit F. On le ferme
On ouvre I
 $D(A,I) \leq d(A,F) + d(F,I) \leq 415$
Ouverts = {E,C,I}

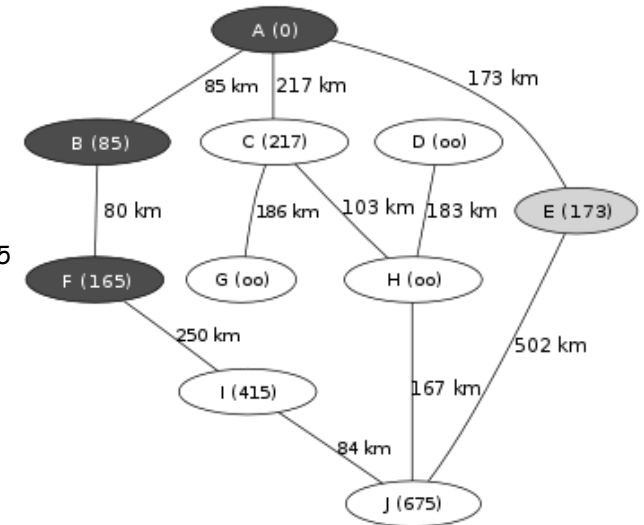


Théorie des Graphes - 2015/2016

Chemin de A vers J

146

On choisit E. On le ferme
On ouvre J
 $D(A,J) \leq d(A,E) + d(E,J) \leq 675$
Ouverts = {C,I,J}

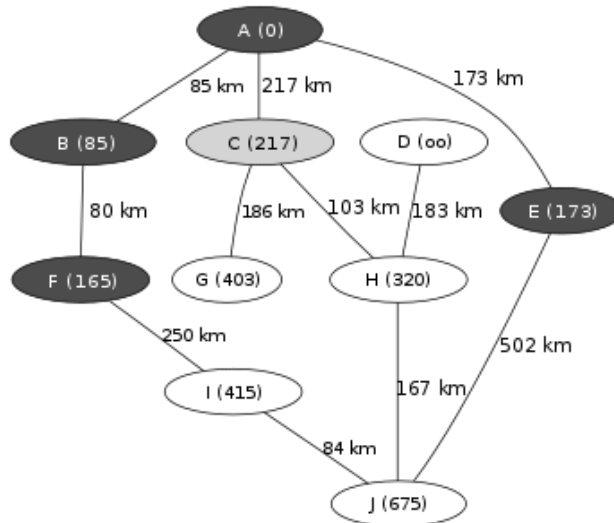


Théorie des Graphes - 2015/2016

Chemin de A vers J

147

On prend C
On ouvre G et H
Ouverts = {H,G,I,J}

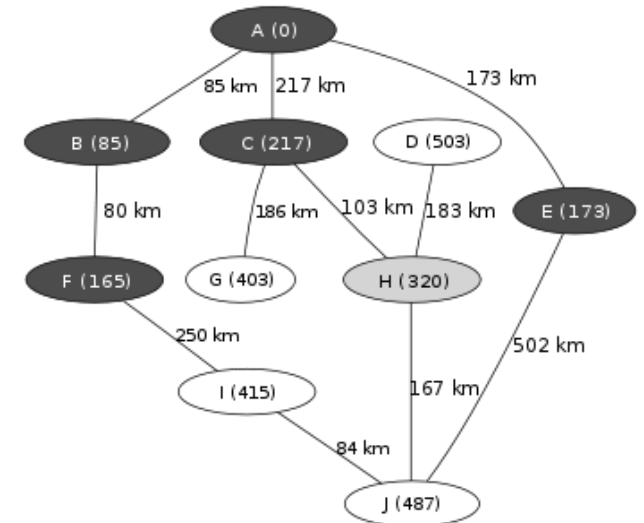


Théorie des Graphes - 2015/2016

Chemin de A vers J

148

On prend H
On met à jour J

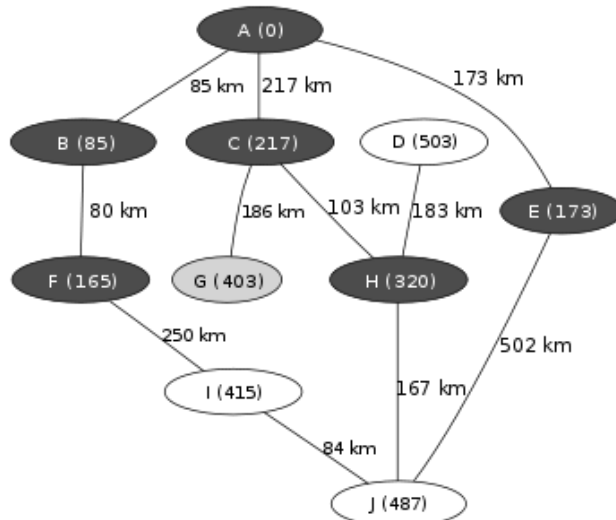


Théorie des Graphes - 2015/2016

Chemin de A vers J

149

On prend G

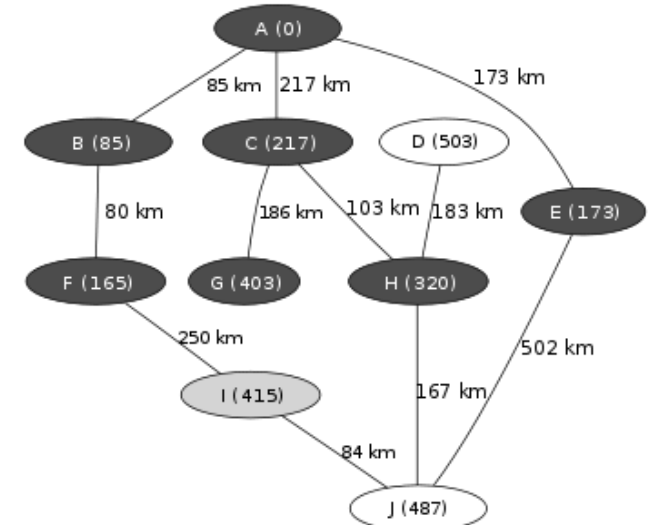


Théorie des Graphes - 2015/2016

Chemin de A vers J

150

On prend I

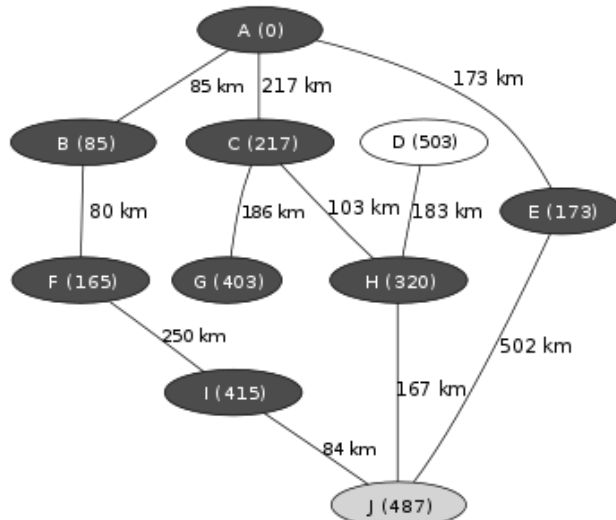


Théorie des Graphes - 2015/2016

Chemin de A vers J

151

On prend J
Fin de l'algorithme



Théorie des Graphes - 2015/2016

Dijkstra

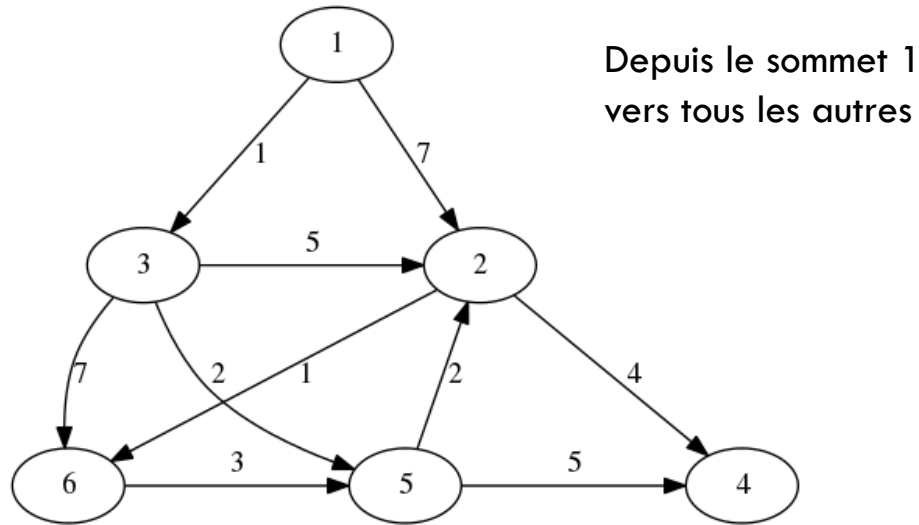
152

- maj_distances(s1,s2)
 si $d[s2] > d[s1] + \text{longueur}(s1,s2)$
 alors $d[s2] \leftarrow d[s1] + \text{longueur}(s1,s2)$
- Dijkstra(G,Poids,sdeb)
 Ouvert $\leftarrow \{s\}$
 tant que Ouvert $\neq \emptyset$ et t n'est pas fermé
 faire i \leftarrow Trouve_min(Ouvert)
 supprimer i de Ouvert
 fermer(i)
 pour chaque nœud k voisin de i
 faire si k n'est pas fermé
 alors ajouter k dans Ouvert s'il n'y était pas déjà
 maj_distances(i,k)
 fait
- Pour trouver le chemin il faut garder le predecesseur (celui qui a mis à jour la distance min)

Théorie des Graphes - 2015/2016

Exercice 1 : plus court chemin

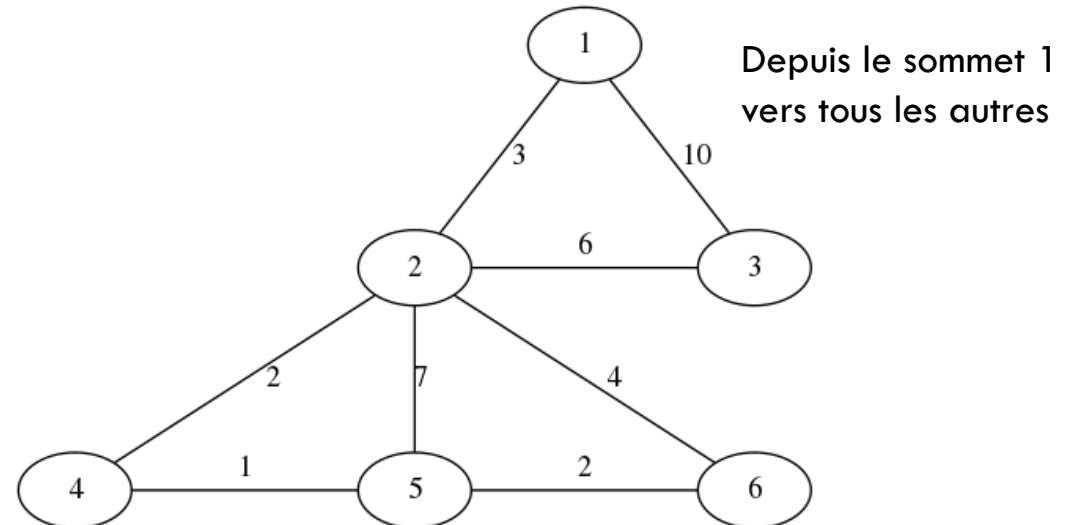
153



Théorie des Graphes - 2015/2016

Exercice 2 : plus court chemin

154



Théorie des Graphes - 2015/2016

Algorithme de Dijkstra

155

- La difficulté vient de la détermination du sommet situé à la plus petite distance
 - ▣ Queue de Priorité

Théorie des Graphes - 2015/2016

Queues de priorité : rappels

156

- 3 opérations
 - ▣ insert(elt,val)
 - ▣ decreaseKey(elt,val)
 - ▣ extractMin()
- Monotone ou pas (l'élément extrait augmente ou pas)
- Implémentations (complexité ?)
 - ▣ Liste
 - ▣ Liste ordonnée
 - ▣ Tas
 - ▣ Buckets

Théorie des Graphes - 2015/2016

Queues de priorité : rappels

157

- Bucket : on crée un tableau de C valeurs possible (c'est le max du cout d'une arete de s à t)
- On traverse le tableau pour trouver le plus petit élément
- On utilise le tableau de façon circulaire
- Pourquoi est-ce que cela marche ?
- Complexité $O(m+nC)$

Théorie des Graphes - 2015/2016

Queues de priorité

158

- Il existe de nombreuses structures de données
 - ▣ K-ary heap
 - ▣ Fibonacci heap
 - ▣ K-level buckets
 - ▣ Hot queue

Théorie des Graphes - 2015/2016

Propriétés

159

- Tout sous-chemin d'un plus court chemin est un plus courts chemin
- L'ensemble des arcs des plus courts chemins forment un arbre
- La valeur des nœuds extrait par Dijkstra est croissante
- Ce dernier point n'est plus vrai si on a des coûts négatifs

Théorie des Graphes - 2015/2016

Problèmes

160

- Connexité et sommets atteignables
- Plus court chemin entre deux sommets
 - ▣ Algorithme de Dijkstra
 - ▣ **Algorithme de Bellman-Ford**
 - ▣ Combinaison des deux
- Détection de cycles négatifs
- Plus courts chemins entre tous les sommets
- Cas des graphes sans circuit

Théorie des Graphes - 2015/2016

Algorithme de Bellman-Ford

161

- Propriété des chemins. d : distance depuis s
condition d'optimalité
 - $\forall (u,v) \in A, d(v) \leq d(u) + c(u,v)$
- Signification intuitive
 - Soit le plus court chemin de s à v peut passer par l'arc (u,v) et dans ce cas on a : $d(v) = d(u) + c(u,v)$
 - Soit le plus court chemin de s à v ne peut pas passer par l'arc (u,v) et dans ce cas on a : $d(v) < d(u) + c(u,v)$

Théorie des Graphes - 2015/2016

Algorithme de Bellman-Ford

162

- Principe :
 - On met à jour les distances jusqu'à vérifier la propriété précédente pour chaque arc
 - Quand on trouve un arc (u,v) avec $d(v) > d(u) + c(u,v)$, cela signifie qu'on peut réduire la distance de s à v en passant par u .
 - On met donc $d(v)$ à jour en passant par u
 - $d(v) \leftarrow d(u) + c(u,v)$,
 - On commence avec $d(s)=0$ et ∞ pour tous les autres sommets

Théorie des Graphes - 2015/2016

LabelCorrecting

163

LABELCORRECTING

$d(s) \leftarrow 0$

$support(s) \leftarrow nil$

$\forall i \in N - \{s\} : d(i) \leftarrow \infty$

while $\exists (i,j) s.t. d(j) > d(i) + c_{ij}$ **do**

$d(j) \leftarrow d(i) + c_{ij}$
 $support(j) \leftarrow i$

Théorie des Graphes - 2015/2016

Bellman-Ford

164

- Preuve
 - Pour tout sommet u , $d(u)$ décroît tout au long de l'algorithme
 - En l'absence de cycle négatif, à la fin de l'algorithme on a
 - $\forall i \in X: -nC \leq d(i) \leq nC$
 - Preuve: comme il n'y a pas de cycle négatif un chemin emprunte au plus $n-1$ arcs
 - LabelCorrecting est en $O(n^2C)$
 - Preuve : $d(i)$ peut être modifié au plus $2nC$ fois car il est décrémenter de 1 à chaque fois. Il y a n sommets dont au plus $O(n^2C)$ décréments

Théorie des Graphes - 2015/2016

Bellman-Ford

- Modified LabelCorrecting
- Le problème central est la détection et l'ordre de traitement des arcs (u,v) tels que
 - ▣ $d(v) > d(u) + c(u,v)$
 - ▣ Cela signifie qu'on peut réduire la distance en passant par u
- Plusieurs idées possibles
 - ▣ On met l'ensemble des arcs dans une liste et on les traite les uns après les autres.
On gère l'introduction de nouveaux arcs : ce sont les arcs voisins d'un sommet dont la distance vient d'être modifiée.
 - ▣ On utilise une liste de sommets. Un sommet u appartient à la liste s'il existe un arc (u,v) qui viole la condition d'optimalité

Modified LabelCorrecting

```
MODIFIEDLABELCORRECTING
d(s) ← 0
support(s) ← nil
∀i ∈ N - {s} : d(i) ← ∞
i ← s
do
  for chaque (i, j) ∈ A(i) do
    if d(j) > d(i) + cij then
      d(j) ← d(i) + cij
      support(j) ← i
      ADDNODE(j, Stream)
  i ← SELECTNODE(Stream)
while i ≠ nil
```

Modified LabelCorrecting

- Stream a une structure de FIFO
- On la décompose en 2 ensembles S_1 et S_2
 - ▣ addNode ajoute toujours en fin de S_2
 - ▣ selectNode choisit le premier élément de S_1 si S_1 n'est pas vide; sinon S_2 est copiée dans S_1 et puis vidée. Dans ce cas on parle de **changement de passe**.
- La passe i traite les nœuds qui ont été introduit en $i-1$

Modified LabelCorrecting

BELLMAN-FORD-MOORE

Stream est décomposé en 2 ensembles S_1 et S_2

```
ADDNODE(node, Stream)
└ if node ∉ S2 then ADDLAST(node, S2)
```

```
SELECTNODE(node, Stream)
┌ if S1 = ∅ then
  ┌ if S2 = ∅ then nil
  └ S1 ← S2
  └ S2 ← ∅
node ← FIRSTNODE(S1)
REMOVE NODE(node, S1)
└ node
```

Bellman-Ford

169

□ Théorème

- L'algorithme de Bellman-Ford est tel que
 - Chaque passe est en $O(m)$
 - Le nombre de passes est borné par la hauteur de l'arbre des plus courts chemins
 - L'algorithme est en $O(nm)$

Théorie des Graphes - 2015/2016

Modified LabelCorrecting

170

□ Autre possibilité pour Stream:

- LIFO : l'algorithme n'est plus polynomial ! Mais il fonctionne parfois très bien pratique

□ Remarque :

- **Aucune supposition n'a été faite sur les coûts !**

Théorie des Graphes - 2015/2016

Problèmes

171

- Connexité et sommets atteignables
- Plus court chemin entre deux sommets
 - Algorithme de Dijkstra
 - Algorithme de Bellman-Ford
 - **Combinaison des deux**
- Détection de cycles négatifs
- Plus courts chemins entre tous les sommets
- Cas des graphes sans circuit

Théorie des Graphes - 2015/2016

Bellman-Ford + Dijkstra ?

172

- Si on a uniquement des coûts non négatifs et si on prends l'algorithme Modified LabelCorrecting et que Stream est implémenté comme une queue de priorité avec la distance à s comme valeur, alors
 - On implémente l'algorithme de Dijkstra !
 - Un nœud ne peut-être introduit dans Stream qu'une seule fois

Théorie des Graphes - 2015/2016

Bellman-Ford + Dijkstra ?

173

- Que se passe t'il si Stream est une queue de priorité et si les coûts ne sont pas tous négatifs ?
 - ▣ Bellman-Ford fonctionne toujours (pas de supposition ni sur Stream, ni sur le coût des arcs)
 - ▣ Algorithme de Dinitz et Itzhak (2010).
 - ▣ Complexité en $O(nm)$

Théorie des Graphes - 2015/2016

Plus court chemin de s à t

174

- Si on cherche le plus court chemin de s à t, les algorithmes précédents recherchent un plus court chemin de s à tous les autres nœuds jusqu'à rencontré t
- Est-ce que l'on peut accélérer cela et éviter, dans le cas euclidien, de faire des cercles qui s'agrandissent ?
 - ▣ Si on cherche un plus court chemin entre Lille et Nice, les algos considèrent Amsterdam !

Théorie des Graphes - 2015/2016

Plus court chemin de s à t

175

- Pour éviter cela, on peut utiliser un **minorant** de la distance minimum entre un nœud i et t
- Dès que l'on aura trouver un majorant de la distance de s à t alors on pourra utiliser ces minorants
 - ▣ $d(s,t) \leq M$
 - ▣ Si $d(s,i) + m(i,t) > M$ alors on arrête d'explorer depuis i
- Par exemple:
 - ▣ on sait que Amsterdam est au moins à 1000km de Nice.
 - ▣ on sait que Lille est au plus à 1200 km de Nice
 - ▣ on trouve que $d(\text{Lille}, \text{Amsterdam}) = 280 \text{ km}$
 - ▣ on en déduit que $d(\text{Lille}, \text{Amsterdam}) + d(\text{Amsterdam}, \text{Nice}) > d(\text{Lille}, \text{Nice})$. On arrête l'exploration depuis amsterdam

Théorie des Graphes - 2015/2016

Problèmes

176

- Connexité et sommets atteignables
- Plus court chemin entre deux sommets
 - ▣ Algorithme de Dijkstra
 - ▣ Algorithme de Bellman-Ford
 - ▣ Combinaison des deux
- **Détection de cycles négatifs**
- Plus courts chemins entre tous les sommets
- Cas des graphes sans circuit

Théorie des Graphes - 2015/2016

Détection de cycles négatifs

177

- C = plus grand cout d'un arc.
- Méthode 1 : Si une distance atteint une valeur $\leq -nC$ alors un cycle négatif est présent
- Méthode 2 :
 - ▣ On utilise le Modified LabelCorrecting algorithm avec une FIFO et la décomposition par passe.
 - ▣ On compte pour chaque nœud le nombre de fois où il est introduit dans la queue.
 - ▣ Si un nœud est introduit plus de $(n-1)$ fois alors il y a un circuit négatif

Théorie des Graphes - 2015/2016

Détection de cycles négatifs

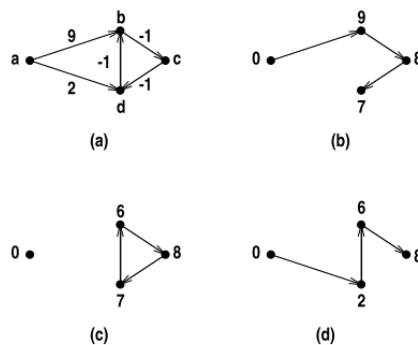
178

- Méthode 3 :
 - ▣ Quand on modifie la distance d'un nœud v : $d(v) \leftarrow d(u) + c(u,v)$. On dit que u est le prédécesseur de v , on le note $p(v)$
 - ▣ Le graphe des prédécesseurs est le sous graphe engendré par les arcs $(p(v),v)$
 - ▣ Si le graphe des prédécesseurs a un cycle alors G contient un cycle négatif.

Théorie des Graphes - 2015/2016

Détection de cycles négatifs

179



Théorie des Graphes - 2015/2016

Détection de cycles négatifs

180

- Méthode 4 : on applique la méthode 3 de temps en temps

Théorie des Graphes - 2015/2016

Détection de cycles négatifs

181

- Méthode 5 : on applique la méthode 3 mais au lieu de considérer la graphe des prédécesseurs, on considère le graphe des arcs admissibles :
 - ▣ Arc admissible : arc (u,v) tel que $d(v) \geq d(u) + c(u,v)$

Théorie des Graphes - 2015/2016

Détection de cycles négatifs

182

- Méthode 6 : Subtree traversal
 - ▣ Quand on applique l'opération de mise à jour de la distance à un arc (u,v) on crée un cycle dans le graphe des parents ssi u est un ancêtre de v
 - ▣ On vérifie cela lors de la mise à jour de la distance
- Méthode 7 : Tarjan subtree disassembly (dépasse le cadre de ce cours)

Théorie des Graphes - 2015/2016

Problèmes

183

- Connexité et sommets atteignables
- Plus court chemin entre deux sommets
 - ▣ Algorithme de Dijkstra
 - ▣ Algorithme de Bellman-Ford
 - ▣ Combinaison des deux
- Détection de cycles négatifs
- **Plus courts chemins entre tous les sommets**
- Cas des graphes sans circuit

Théorie des Graphes - 2015/2016

Plus courts chemins entre tous les sommets

184

- $\forall i,j$: on veut connaître la distance du plus court chemin entre i et j
- 2 méthodes principales
 - ▣ Repeated shortest path problem
 - ▣ Algorithme de Floyd Warshall

Théorie des Graphes - 2015/2016

Repeated shortest path

185

- Idée simple : on répète l'algo de plus court chemin n fois
- A partir d'un nœud i : on calcule tous les plus courts chemins de i vers t à l'aide d'un algorithme (Dijkstra ou Belmann Ford)
- Si on répète n fois on a répondu au problème

Elimination des couts négatifs

186

- Introduction d'une technique importante
 - ▣ **Les coûts réduits**
- Attention : beaucoup de livres introduisent cette notion sans donner son intérêt et en faisant comme si elle était obligatoire. Ce n'est pas vrai. Elle est utile mais c'est tout.

Coûts réduits

187

- Supposons que l'on ait calculé les distances d entre s et tous les sommets.
- Pour chaque arc (u,v) on peut calculer la valeur
 - ▣ $c^d(u,v) = c(u,v) + d(u) - d(v)$
 - ▣ C'est le coût réduit de l'arc (u,v)
 - ▣ Cela représente le surcoût local à passer par cet arc
- On a toujours $c^d(u,v) \geq 0$
 - ▣ La condition d'optimalité $d(v) \leq d(u) + c(u,v)$ implique $0 \leq c(u,v) + d(u) - d(v)$, d'où $c^d(u,v) \geq 0$

Coûts réduits

188

- On peut remplacer les coûts initiaux par les coûts réduits
- En procédant ainsi il ne reste que des coûts positifs
- On peut ainsi appliquer n fois l'algorithme de Dijkstra
- On obtient les valeurs d'origine des distance en appliquant la transformation suivante pour chaque paire de noeud u et v : on ajoute $d(v) - d(u)$ à la distance calculée dans le graphe contenant les coûts réduits

Algorithme de Floyd Warshall

189

- Algorithme de programmation dynamique en $O(n^3)$.
- Basé sur le théorème suivant :
Les distances sont optimales ssi
 $\forall u,v,w \ d[u,v] \leq d[u,w] + d[w,v]$
- Appelons $d^k[u,v]$ la longueur du plus court chemin utilisant au maximum les nœuds 1 à $k-1$. On a
 - $d^{k+1}[u,v] = \min \{d^k[u,v], d^k[u,k] + d^k[k,v]\}$

Théorie des Graphes - 2015/2016

Algorithme de Floyd Warshall

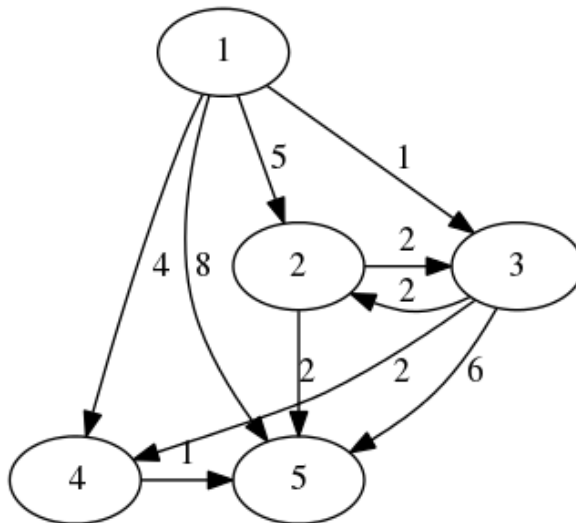
190

- $\forall u,v \ d[u,v] \leftarrow \infty; \text{pred}[u,v] \leftarrow \text{null}$
 $\forall u \ d[u,u] \leftarrow 0$
 $\forall (u,v) \in A \ d[u,v] \leftarrow c(u,v); \text{pred}[u,v] \leftarrow v$
 Pour k de 1 à n
 Pour u de 1 à n
 Pour v de 1 à n
 Si $d[u,v] > d[u,k] + d[k,v]$
 Alors $d[u,v] \leftarrow d[u,k] + d[k,v]$
 $\text{pred}[u,v] \leftarrow k$
 Fin si
 Fin pour
 Fin pour
 Fin pour

Théorie des Graphes - 2015/2016

Exercice : Floyd Warshall

191



Théorie des Graphes - 2015/2016

Problèmes

192

- Connexité et sommets atteignables
- Plus court chemin entre deux sommets
 - Algorithme de Dijkstra
 - Algorithme de Bellman-Ford
 - Combinaison des deux
- Détection de cycles négatifs
- Plus courts chemins entre tous les sommets
- **Cas des graphes sans circuit**

Théorie des Graphes - 2015/2016

Graphes sans circuits

Théorie des Graphes - 2015/2016

Graphes sans circuits

194

- Attention ce ne sont pas nécessairement des arbres.
- On parle ici de circuit et non pas de cycle.
- On est donc dans le cas ORIENTE
- Les problèmes du plus court chemin et de plus long chemin deviennent facile
- On calcule d'abord l'ordre topologique du graphe

Théorie des Graphes - 2015/2016

Fonction rang d'un graphe

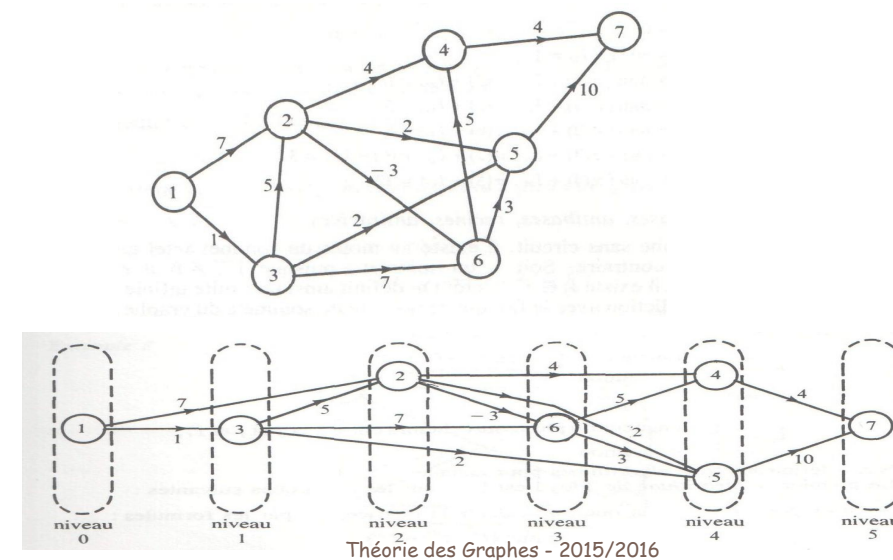
195

- Ordre topologique (un sommet est toujours visité avant ses successeurs)
- Associe à chaque nœud i un nombre positif $\text{rang}(i)$ tel que
 - ▣ $r(1)=0$
 - ▣ $R(i)$ = nombre d'arcs dans un chemin de cardinalité maximum entre 1 et i (taille (nombre de sommets) du plus grand chemin de prédécesseurs depuis 1)
- Se calcule facilement : on élimine les sommets sans prédécesseur à chaque étape (on épluche le graphe)

Théorie des Graphes - 2015/2016

Rang d'un graphe

196



Théorie des Graphes - 2015/2016

Rang d'un graphe

197

- **TraitementNiveau(Niveau k, R) :**
Pour chaque élément i de R faire
 - $S \leftarrow$ vide
 - $\text{Rang}(i) \leftarrow k$
 - Pour chaque voisin j de i faire
 - Décrémenter $\text{degré}(j)$
 - Si $\text{degré}(j) = 0$ alors ajouter j dans S $R \leftarrow S$
- **Algorithme :**
 - On détermine les racines. On les place dans R
 - $\text{Niveau} \leftarrow 0$
 - Répéter tant que R n'est pas vide
 - **TraitementNiveau(Niveau, R)**
 - Incréments Niveau

Théorie des Graphes - 2015/2016

Chemins et graphes sans circuit

198

- On prend les sommets selon l'ordre topologique
- Plus court chemin jusqu'au nœud v
 - Minimum des plus courts chemins à partir de ses arcs entrants
- Plus long chemin jusqu'au nœud v
 - Maximum des plus longs chemins à partir de ses arcs entrants

Théorie des Graphes - 2015/2016

Problème d'ordonnancement

199

- Un **problème d'ordonnancement** est composé de façon générale d'un ensemble de tâches soumises à certaines contraintes, et dont l'exécution nécessite des ressources.
- **Résoudre un problème d'ordonnancement** consiste à **organiser ces tâches**, c'est-à-dire à déterminer leurs dates de démarrage et d'achèvement, et à leur **attribuer des ressources**, de telle sorte que les contraintes soient respectées.
- Les problèmes d'ordonnancement sont très variés.

Théorie des Graphes - 2015/2016

Paramètres en ordonnancement

200

- tâches
 - morcelables ou non
 - indépendantes ou non
 - durées fixes ou non
- ressources
 - renouvelables ou consommables
- contraintes portant sur les tâches
 - contraintes temporelles
 - fenêtres de temps
- aux critères d'optimalité
 - liés au temps (délai total, délai moyen, retards)
 - aux ressources (quantité utilisée, taux d'occupation)
 - ou à d'autres coûts (production, lancement, stockage)

Théorie des Graphes - 2015/2016

Méthode PERT/CPM

201

- **PERT (Programme Evaluation Review Technique)** introduit par l'armée américaine (navy) en 1958 pour contrôler les coûts et l'ordonnancement de la construction des sous-marins Polaris.
- **CPM (Critical Path Method)** introduit par l'industrie américaine en 1958 (DuPont Corporation and Remington-Rand) pour contrôler les coûts et l'ordonnancement de production.
- **PERT/CPM ignore la plupart des dépendances.**

Théorie des Graphes - 2015/2016

Intérêts de PERT/CPM

202

- Quel sera la durée totale du projet ?
- Quelles sont les risques ?
- Quelles sont les **activités critiques** qui perturberont l'ensemble du projet si elles ne sont pas achevées dans les délais ?
- Quel est le statut du projet ? En avance ? Dans les temps ? En retard ?
- Anticiper les conséquences des retards et dépassements.
- Si la fin du projet doit être avancée, quelle est la meilleure manière de procéder ?

Théorie des Graphes - 2015/2016

Principe de PERT/CPM

203

- On suppose que le projet se décompose en tâches
- Chaque tâche i est caractérisée par
 - ▣ Sa durée $d(i)$
 - ▣ Ses contraintes d'antériorité avec les autres tâches.
- **PERT/CPM ignore la plupart des dépendances :**
 - ▣ partage de ressources pour la réalisation des tâches.

Théorie des Graphes - 2015/2016

Graphe potentiels-tâches

204

- A chaque tâche i on associe un sommet du graphe
- Si la tâche i doit précéder la tâche j alors on définit un arc (i,j) de longueur $d(i)$ (durée de i)
- On ajoute deux sommets fictifs
 - ▣ α qui correspond à la tâche de début des travaux (durée 0) qui est antérieure à toutes les autres tâches
 - ▣ ω qui correspond à la tâche de fin des travaux (durée 0) qui est postérieure à toutes les autres tâches
- Pas de circuit dans le graphe : sinon problème i doit suivre j , k doit suivre i et j doit suivre k : blocage

Théorie des Graphes - 2015/2016

Graphe potentiels-tâches

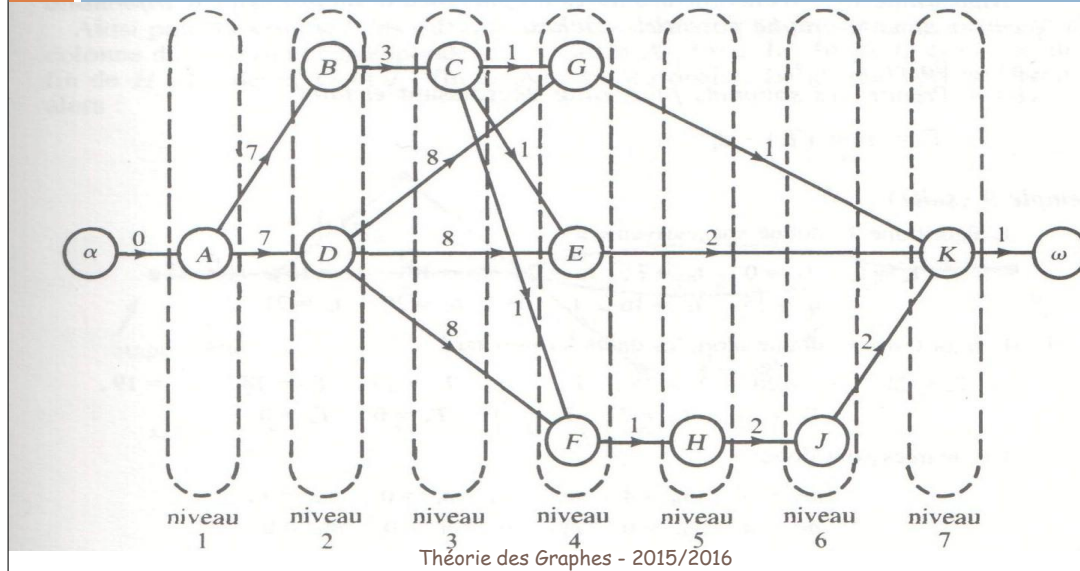
205

Code de la tâche		Durée (semaine)	Tâches antérieures
A	Maçonnerie	7	-
B	Charpente	3	A
C	Toiture	1	B
D	Installations sanitaires et élec.	8	A
E	Façades	2	D,C
F	Fenêtres	1	D,C
G	Jardin	1	D,C
H	Plafonnage	2	F
J	Peinture	2	H
K	Emménagement	1	E,G,J

Théorie des Graphes - 2015/2016

Graphe potentiels-tâches

206



Théorie des Graphes - 2015/2016

Ordonnancement

207

- **But** : trouver un ordonnancement qui minimise le délai total du travail, donc la date de fin des travaux
- Pour qu'une tâche puisse commencer, il est nécessaire que toutes les tâches qui la précède (relie à la tâche début) soient réalisées

Théorie des Graphes - 2015/2016

Ordonnancement

208

- **Date de début au plus tôt** de la tâche i
 - $t(i) = \max (t(k) + d(k), \text{ avec } k \text{ prédécesseur de } i)$
 - Plus long chemin de α à i
- **Durée minimale** du projet ($t(\omega)$) = plus long chemin de α à ω

Théorie des Graphes - 2015/2016

Ordonnancement

209

- On fixe à $t(\omega)$ la durée du projet
- **Date de début au plus tard** de la tâche i
 - ▣ $T(i) = \min (T(k) - d(i), \text{avec } k \text{ successeur de } i)$
 - ▣ $T(i) = t(\omega)$ - plus long chemin de i à ω
- **Marge** de la tâche i
 - ▣ Différence entre début au plus tard et début au plus tôt
 - ▣ $T(i) - t(i)$
- **Tâche critique** : tâche dont la marge est nulle

Théorie des Graphes - 2015/2016

Chemin critique

210

- **Tâche critique** :
 - ▣ tâche dont la marge est nulle
 - ▣ tâche sur un plus long chemin de α à ω
- **Il existe au moins un chemin critique.**
- **Attention, il peut exister plusieurs chemins critiques !**
- Les activités de ce chemin déterminent à elles seules la date de fin du projet.
- Il est nécessaire de réduire la longueur de tous les chemins critiques pour avancer la fin du projet.
- Les activités non critiques n'influencent pas la date de fin.

Théorie des Graphes - 2015/2016

Tâche fondamentale

211

- une **étape clé** durant l'accomplissement du projet.
- On doit y porter :
 - ▣ une attention approfondie et
 - ▣ et contrôler rigoureusement sa réalisation.
- Par exemple, une tâche appartenant à tous les chemins critiques.

Théorie des Graphes - 2015/2016

Intérêt pratique de PERT/CPM

212

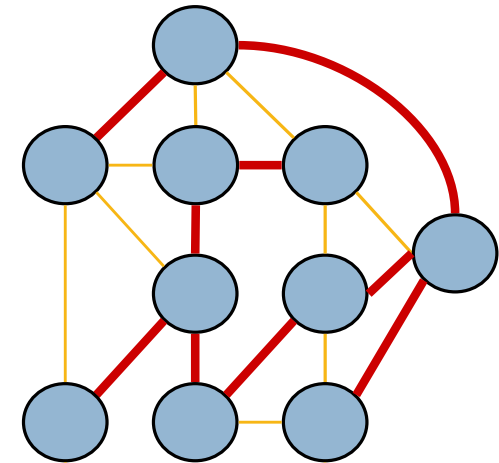
- Ensuite, l'affectation des personnes et ressources peut être améliorée en se concentrant sur ces activités critiques.
- Réciproquement, on peut alors réordonnancer les activités non-critiques et libérer les ressources qui leurs sont allouées sans perturber le reste du projet.
- Gestion basique des incertitudes.

Théorie des Graphes - 2015/2016

Arbre couvrant de poids minimum

Spanning Tree

214



Minimum Spanning Tree

215

- Given a graph G , find a spanning tree where total cost is minimum.

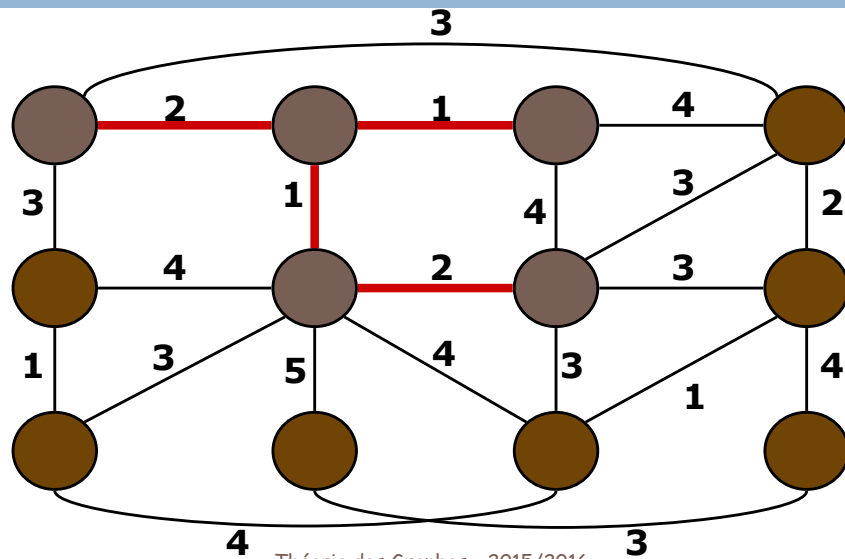
Algorithme de Prim

216

- On utilise un ensemble S de nœuds. On place un seul nœud dedans (celui que l'on veut)
- On ajoute petit à petit dans S tous les nœuds du graphe
- A chaque étape on ajoute dans S un nœud du graphe qui n'est pas dans S et qui est relié à un nœud de S et tel que le coût minimum du lien qui le relie à S est le plus petit possible. On place ce lien minimum dans l'arbre

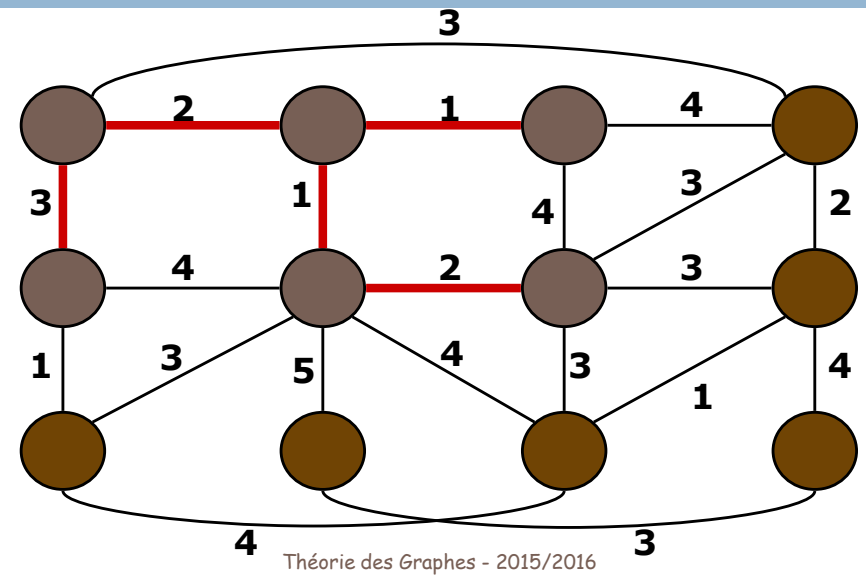
Prim's Algorithm

221



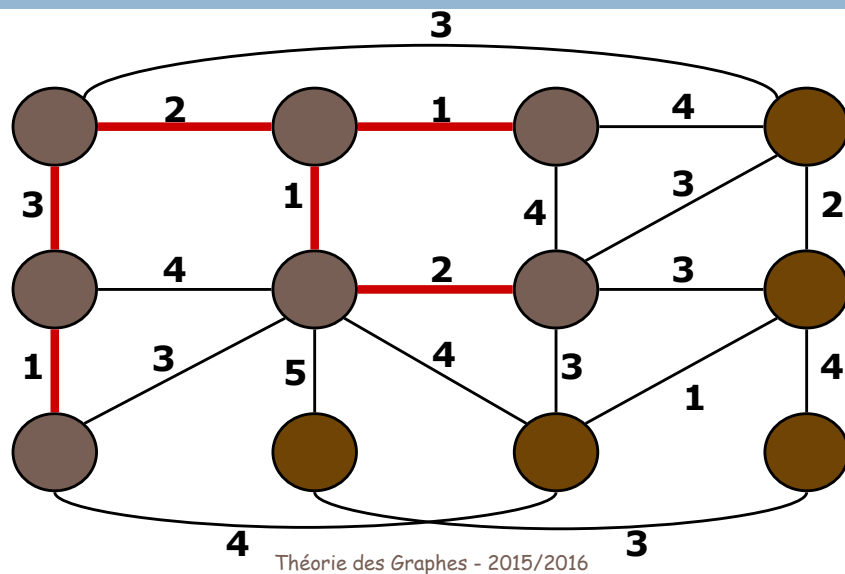
Prim's Algorithm

222



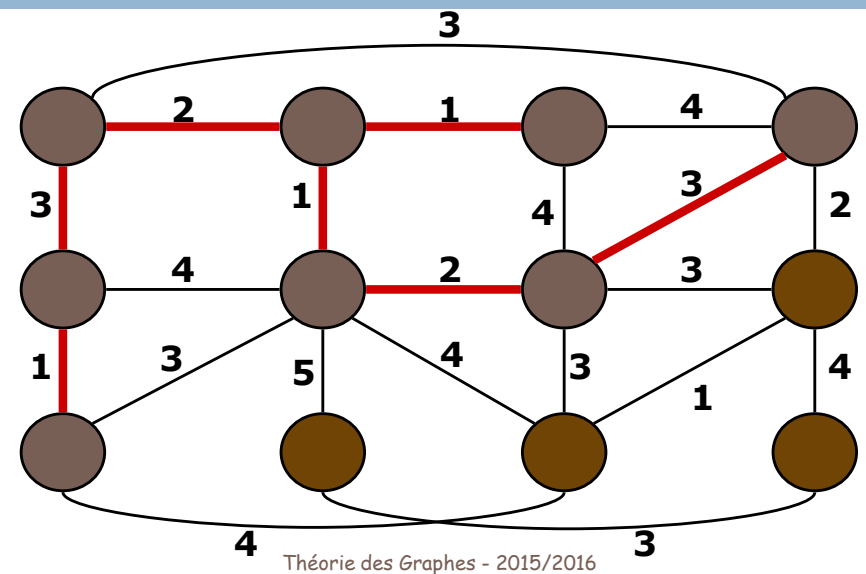
Prim's Algorithm

223



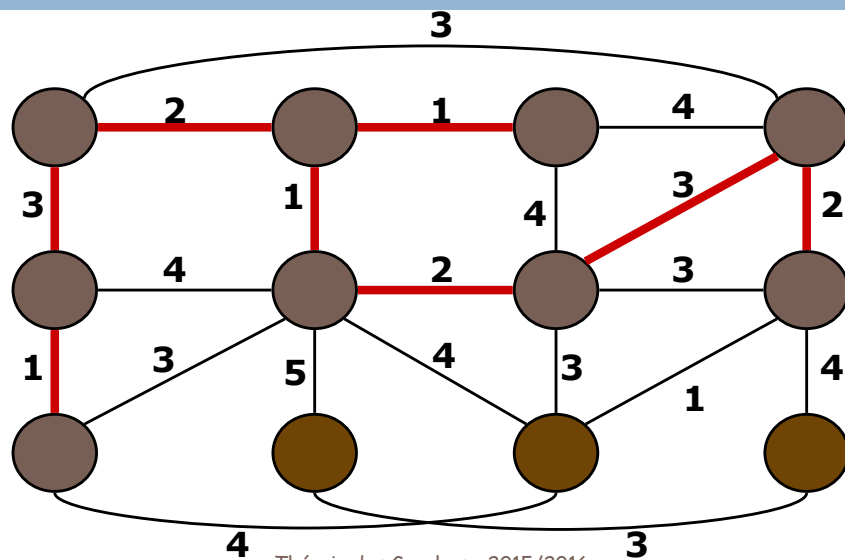
Prim's Algorithm

224



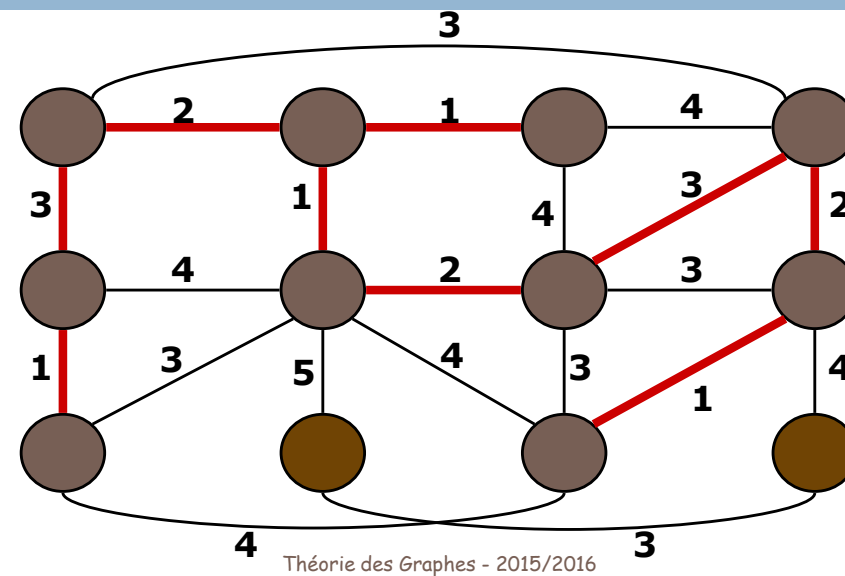
Prim's Algorithm

225



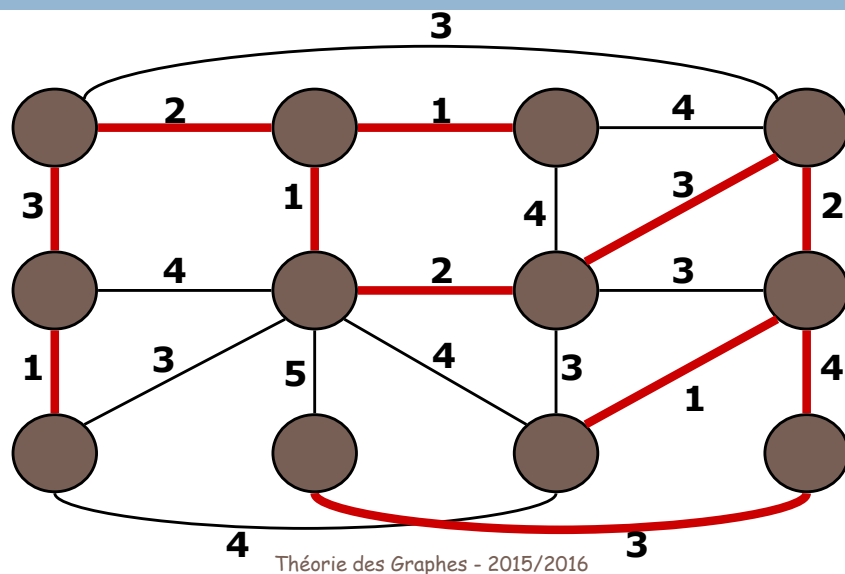
Prim's Algorithm

226



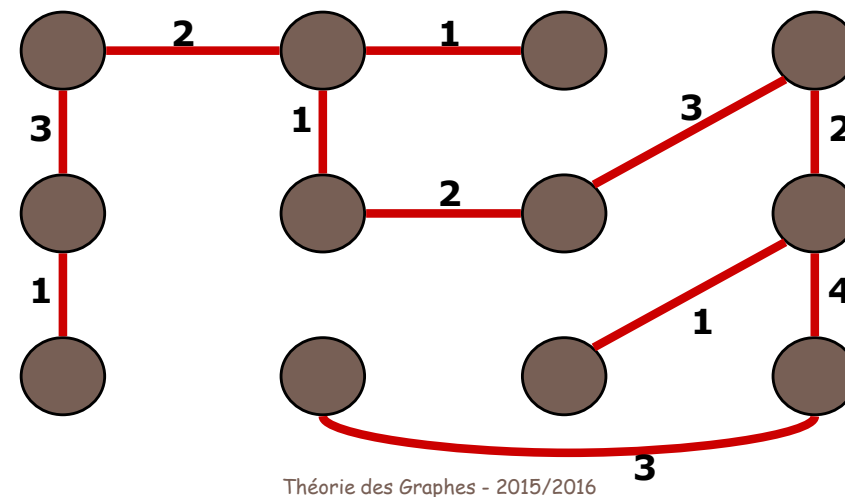
Prim's Algorithm

227



Prim's Algorithm

228



Prim's Greedy Algorithm

```
color all vertices yellow
```

color the root red

while there are yellow vertices

pick an edge (u,v) such that

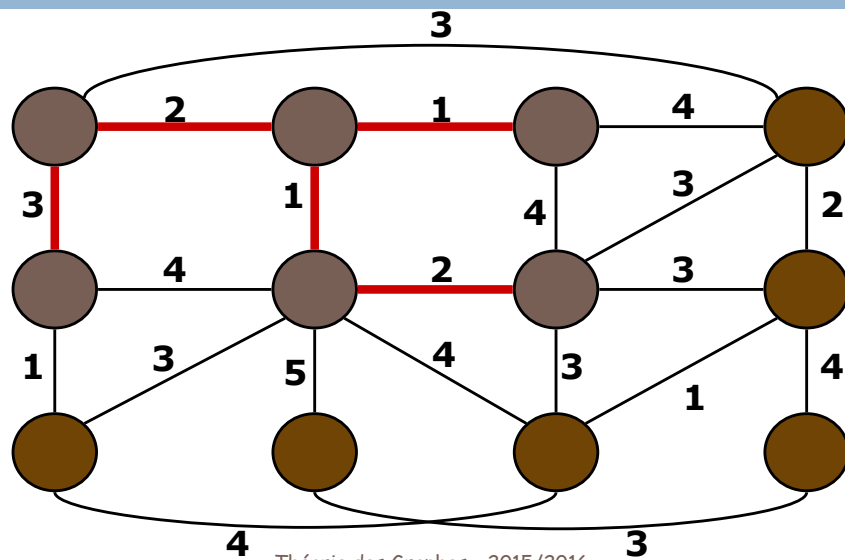
u is red, v is yellow & $\text{cost}(u,v)$ is min

color v red

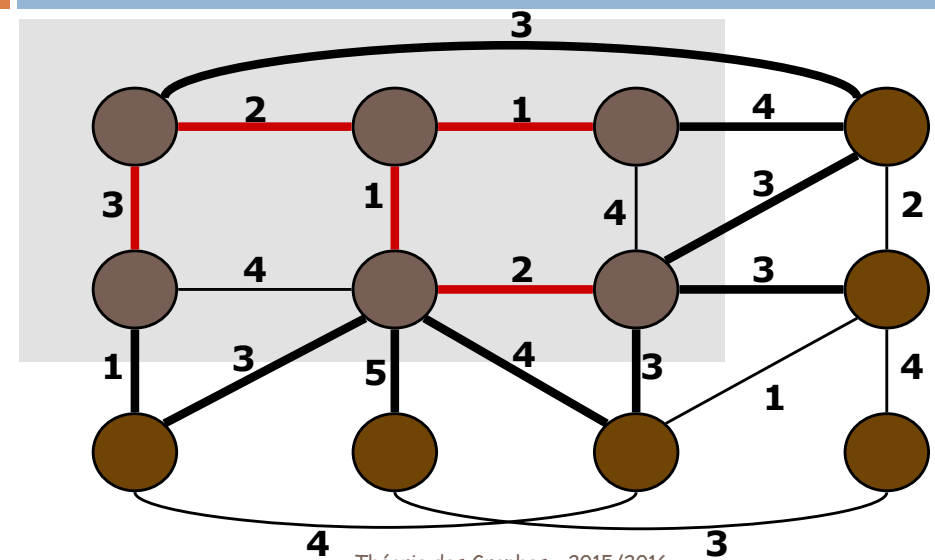
Preuve

- Considérons
 - SG : la solution calculée par l'algorithme glouton
 - Sopt : la solution optimale
- Si $\text{val}(\text{SG}) = \text{val}(\text{Sopt})$ alors pas de problèmes
- Si $\text{val}(\text{SG}) \neq \text{val}(\text{Sopt})$ alors les deux arbres diffèrent. Prenons le premier arc dans l'ordre du Greedy qui diffère entre SG et Sopt. Appelons le e. Quand cet arc a été choisi, c'était le plus petit de la coupe (V, V') . Cet arc relie deux sommets x et y. Dans Sopt x et y sont reliés par un chemin. Sur ce chemin il y a un arc f qui relie un sommet de V à un sommet de V' . Comme l'algo Greedy a préféré e à f : on a $\text{poids}(e) \leq \text{poids}(f)$.
- Prenons la solution Sopt et remplaçons f par e, on obtient Spot'. Si Sopt est optimal alors cela ne peut pas améliorer $\text{val}(\text{Sopt})$. Donc on a $\text{val}(\text{Spot}') = \text{val}(\text{Sopt})$ et donc $\text{poids}(e) = \text{poids}(f)$. On peut reprendre le point précédant avec Sopt' au lieu de Sopt.
- A la fin on va donc montrer qu'on ne peut jamais améliorer Sopt' et que l'arbre construit par le Greedy a la même valeur

Why Greedy Works?

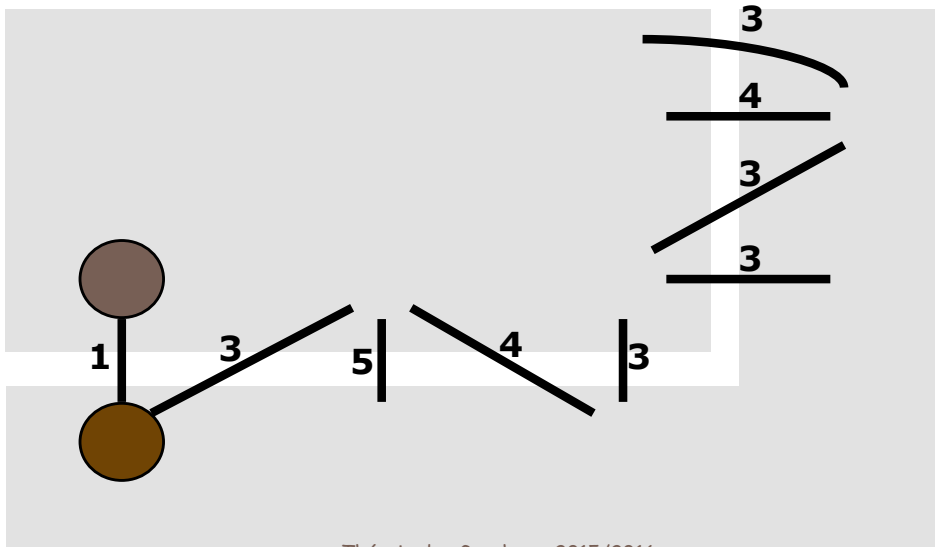


Why Greedy Works?



Why Greedy Works?

233



Théorie des Graphes - 2015/2016

Prim's Algorithm

234

```

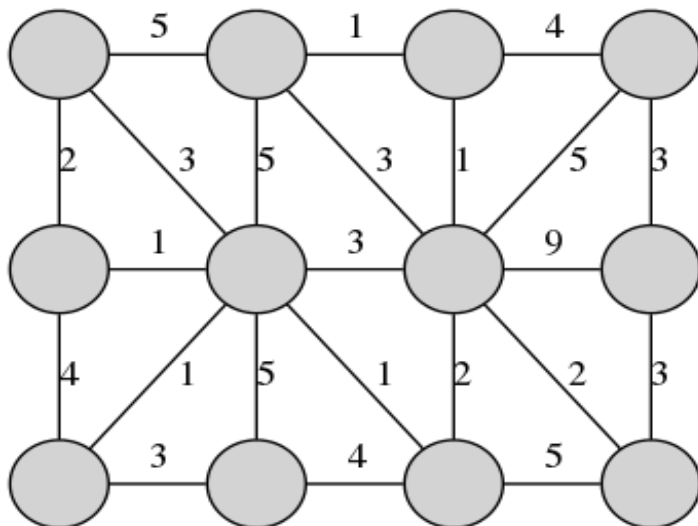
foreach vertex  $v$ 
     $v.key = \infty$ 
 $root.key = 0$ 
 $pq = \text{new PriorityQueue}(V)$  // add each vertex in the priority queue
while  $pq$  is not empty
     $v = pq.deleteMin()$ 
    foreach  $u$  in  $adj(v)$ 
        if  $u$  is in  $pq$  and  $cost(v,u) < u.key$ 
             $pq.decreaseKey(u, cost(v,u))$ 
    
```

Complexity: $O((V+E)\log V)$

Théorie des Graphes - 2015/2016

Exercice : Algorithme de Prim

235



Théorie des Graphes - 2015/2016

Algorithme de Kruskal

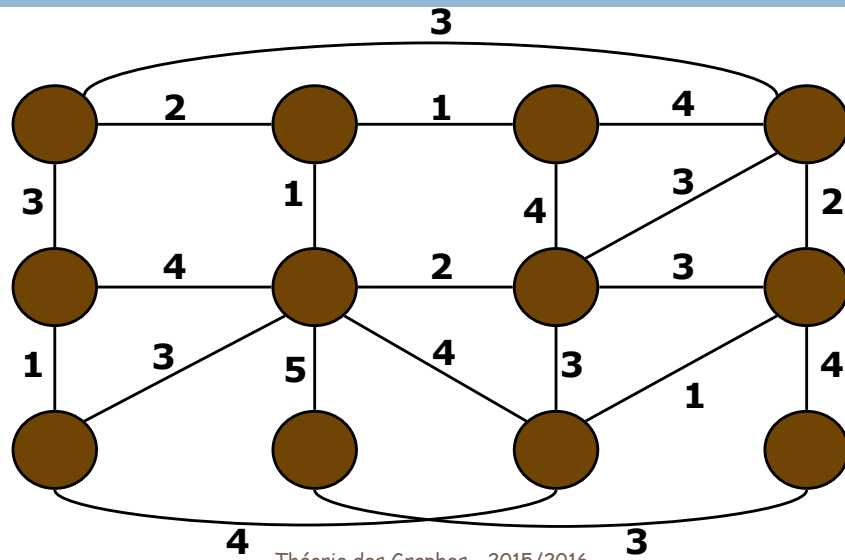
236

- On raisonne avec les arêtes.
- On place chaque nœud dans un ensemble qui lui est propre
- On parcourt les arêtes dans l'ordre croissant des coûts.
 - Si une arête relie deux ensembles disjoints alors on l'ajoute à l'arbre recouvrant et on fusionne les deux ensembles (ils ne sont donc plus disjoints)
 - Si ce n'est pas le cas, on passe à l'arête suivante

Théorie des Graphes - 2015/2016

Kruskal's Algorithm

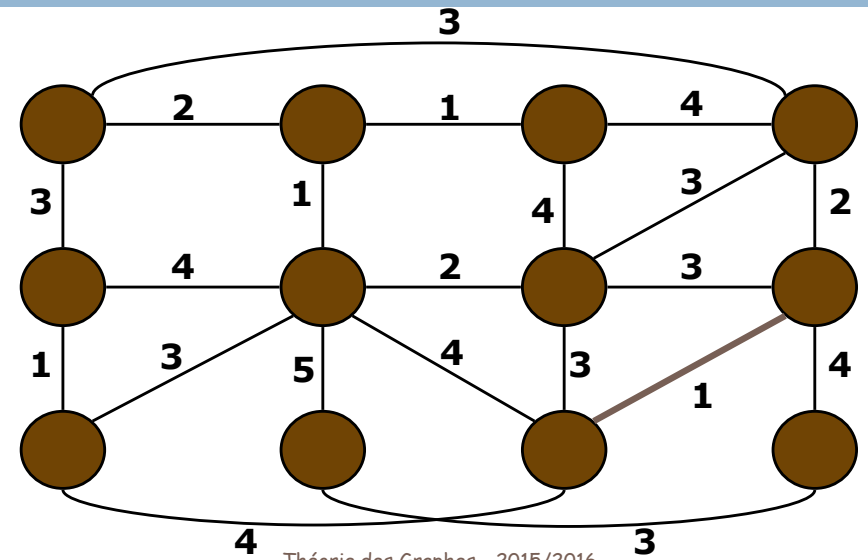
237



Théorie des Graphes - 2015/2016

Kruskal's Algorithm

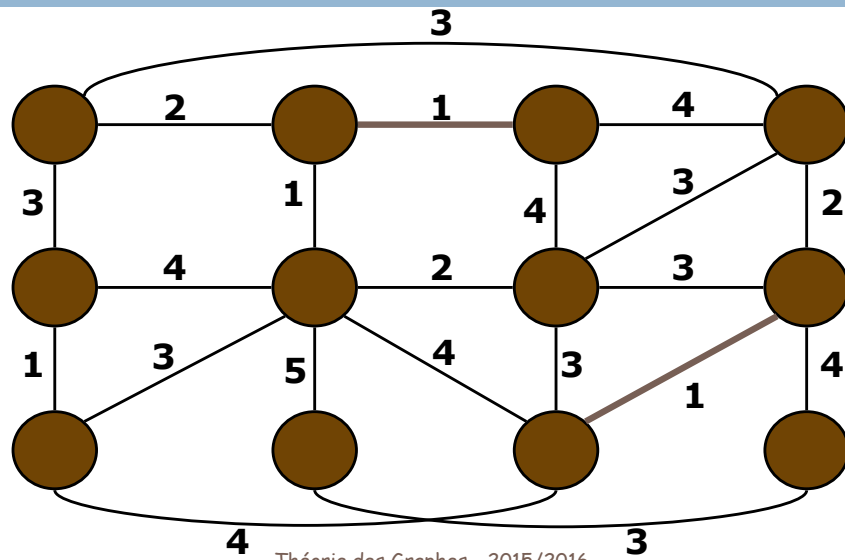
238



Théorie des Graphes - 2015/2016

Kruskal's Algorithm

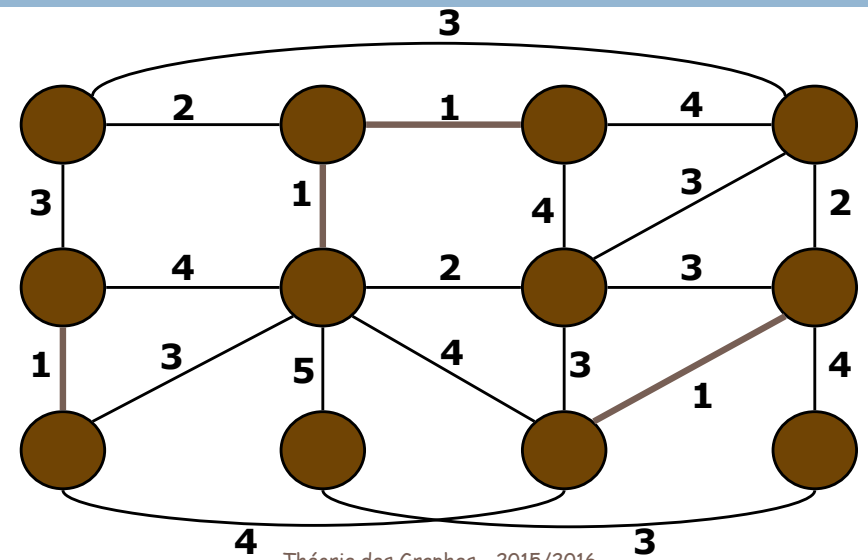
239



Théorie des Graphes - 2015/2016

Kruskal's Algorithm

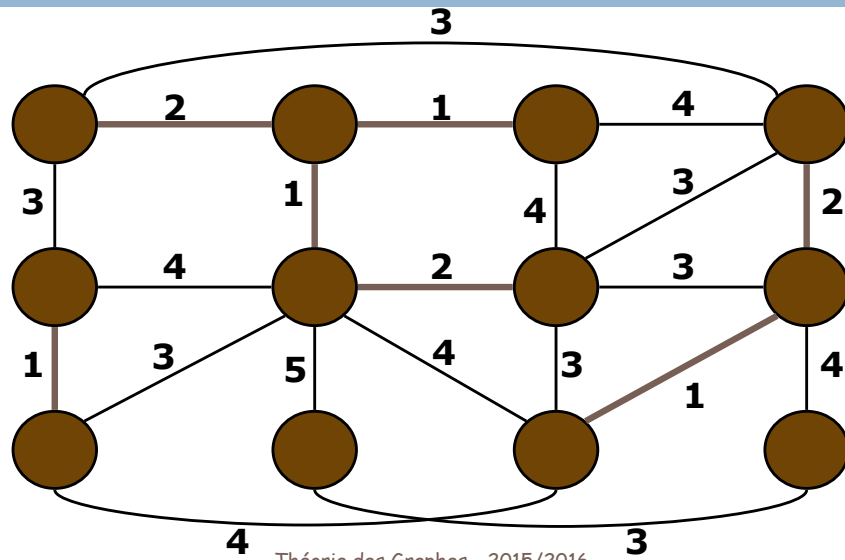
240



Théorie des Graphes - 2015/2016

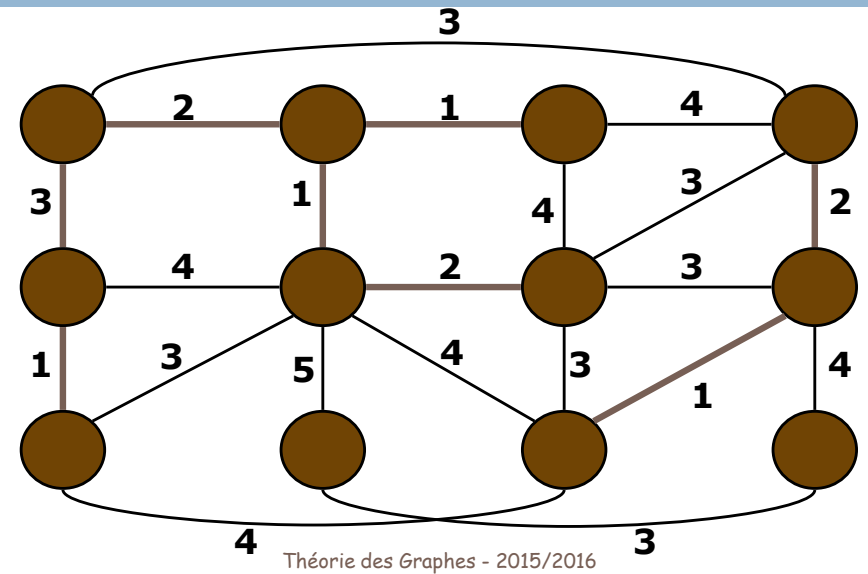
Kruskal's Algorithm

241



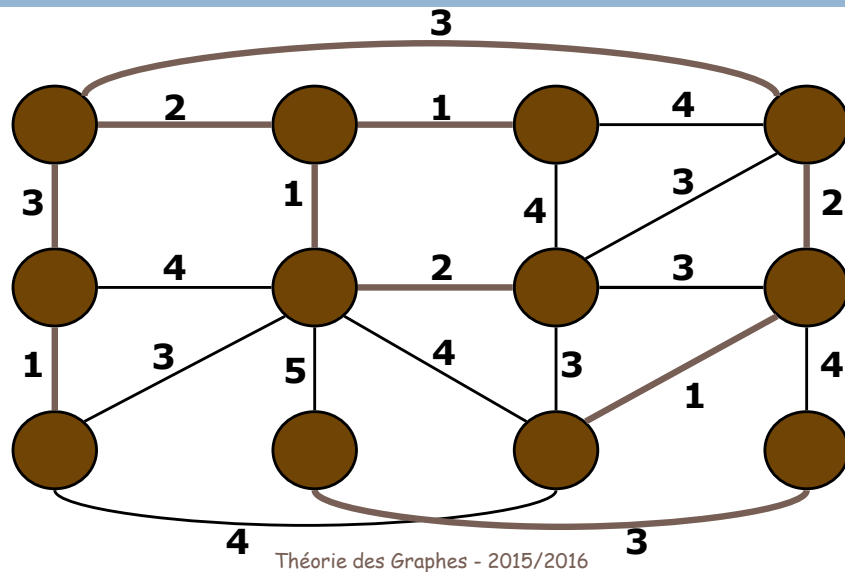
Kruskal's Algorithm

242



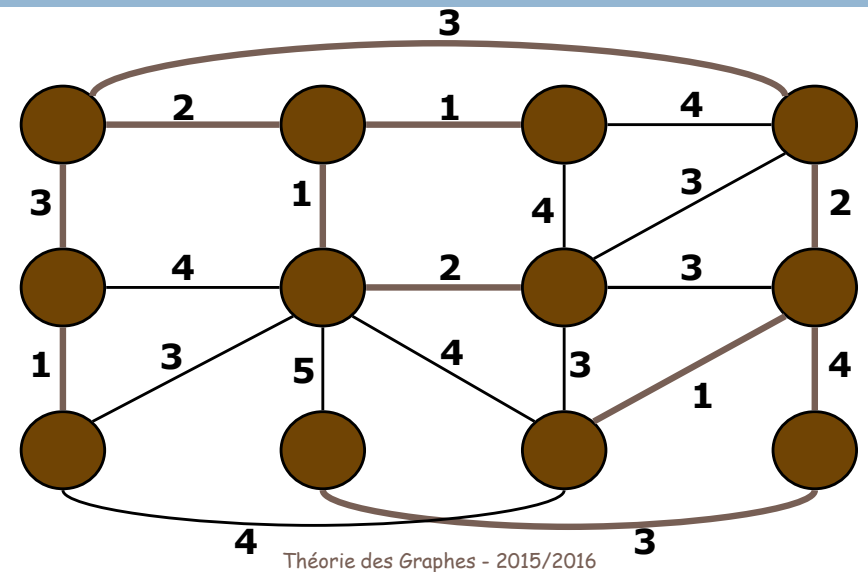
Kruskal's Algorithm

243



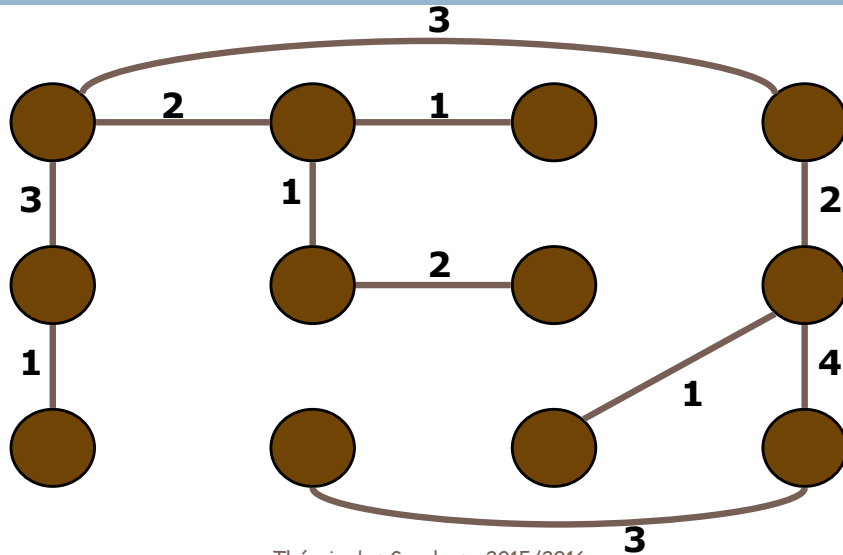
Kruskal's Algorithm

244



Kruskal's Algorithm

245



Théorie des Graphes - 2015/2016

Kruskal's Algorithm

246

```
while there are unprocessed edges left
    pick an edge  $e$  with minimum cost
    if adding  $e$  to MST does not form a cycle
        add  $e$  to MST
    else
        throw  $e$  away
```

Théorie des Graphes - 2015/2016

Preuve

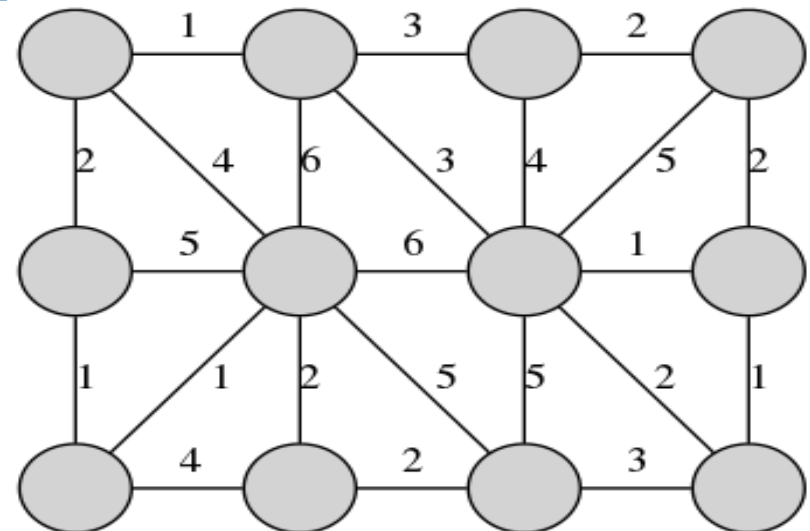
247

- On construit bien un arbre : clairement il ne peut pas y avoir de cycle et s'il y avait deux composantes connexes alors on aurait pris une arête pour les relier
- Minimalité : par induction. Montrons qu'à tout état il existe un arbre recouvrant minimum qui contient les arêtes choisies. F ensemble des arêtes choisies; T arbre recouvrant min contenant F; e prochaine arête choisie. Si e est dans T alors c'est ok. Sinon T+e a un cycle. Soit f l'arête de ce cycle qui est dans T et pas dans F. T - f + e est un arbre recouvrant, son poids est $\geq T$ donc poids f \leq poids de e: poids(f) ne peut pas être $<$ poids(e) sinon on l'aurait choisi. Donc égalité entre les poids et donc F+e appartient à un arbre recouvrant de poids min.

Théorie des Graphes - 2015/2016

Exercice : Algorithme de Kruskal

248



Théorie des Graphes - 2015/2016

Data Structures

249

- How to pick edge with minimum cost?
 - ▣ Sort the array once for all
- How to check if adding an edge can form a cycle?
 - ▣ Use a Disjoint Set

Théorie des Graphes - 2015/2016

Union find data structure

250

- Union find = disjoint set data structure
- Problème à résoudre : on va partitionner des éléments en les regroupant petit à petit. On voudrait savoir
 - ▣ A quel partie un élément appartient ?
 - ▣ Est-ce que deux éléments appartiennent à la même partie ?
- La structure de données Union-Find maintient une partition d'un ensemble et dispose de deux fonctions
 - ▣ Find : trouver (retourne la partie d'un élément)
 - ▣ Union : regroupe deux parties en une

Théorie des Graphes - 2015/2016

Union find

251

- On dispose en outre d'une fonction makeSet qui crée un partie singleton
- Chaque partie (ou ensemble) est identifiée par un élément particulier : son représentant
- **Find(x)** : retourne le représentant de la partie auquel x appartient
- **Union(x,y)** : réunit les deux parties (celle de x et de y) en une seule. Une des deux variable devient le représentant
- **MakeSet(x)** : crée une partie dont x est le représentant

Théorie des Graphes - 2015/2016

Union find : implémentation

252

- Listes chaînées : l'élément en tête est le représentant
 - ▣ MakeSet crée une liste (en $O(1)$)
 - ▣ Union concatène les deux listes (en $O(1)$)
 - ▣ Find : il faut accéder au premier : requiert $O(n)$
- On améliore le Find : chaque élément pointe vers la tête de liste
 - ▣ Cela dégrade l'Union qui devient en $O(n)$...

Théorie des Graphes - 2015/2016

Union find : implémentation

253

- Une bonne solution consiste à utiliser des arbres, représentés à l'aide de la primitive parent
- Le représentant de chaque partie (ensemble) est la racine de l'arbre

Théorie des Graphes - 2015/2016

Union find : implémentation

254

- MakeSet(x)
parent[x] ← null
- Find(x)
si parent[x] = null return x
sinon return Find(parent[x])
- Union(x, y)
xRoot ← Find(x)
yRoot ← Find(y)
parent[xRoot] ← yRoot

Théorie des Graphes - 2015/2016

Union find : implémentation

255

- Dans cette première version, la complexité n'est pas meilleure qu'avec des listes chaînées
- On peut faire deux améliorations importantes
 - ▣ **Union par rang** : On attache l'arbre le plus petit à la racine de l'arbre le plus grand (taille = niveau)
 - ▣ **Compression de chemins** : on met à jour Find quand on l'utilise
Find(x)
si parent[x] ≠ x alors parent[x] ← Find(parent[x])
return parent[x]

Théorie des Graphes - 2015/2016

Union find

256

- Union(x,y)
Lier(Find(x),Find(y))
- Lier(x,y)
si rang[x] > rang[y] alors parent[y] ← x
sinon parent[x] ← y
si rang[x]=rang[y]
rang[y] ← rang[y] + 1

Théorie des Graphes - 2015/2016

Union find : complexité

- On fait m opérations, avec n éléments
- Union pondérée ou par rang est en $O(m + n \log(n))$
- Fonction d'Ackermann :
 - ▣ $A(m,n)=n+1$ si $m=0$
 - ▣ $A(m,n)=A(m-1,1)$ si $m>0$ et $n=0$
 - ▣ $A(m,n)=A(m-1,A(m,n-1))$ si $m > 0$ et $n > 0$

Fonction d'Ackermann

Values of $A(m, n)$						
$m \backslash n$	0	1	2	3	4	n
0	1	2	3	4	5	$n + 1$
1	2	3	4	5	6	$n + 2 = 2 + (n + 3) - 3$
2	3	5	7	9	11	$2n + 3 = 2 \cdot (n + 3) - 3$
3	5	13	29	61	125	$2^{(n+3)} - 3$
4	13	65533	$2^{65536} - 3$	$2^{2^{65536}} - 3$	$2^{2^{2^{65536}}} - 3$	$2^{2^{\dots^2}} - 3$ $n + 3$

Union find : complexité

- On peut montrer que l'on a une borne sup en $O(m \log^*(n))$: voir Cormen-Leiserson-Rivest
- Tarjan a montré que la complexité est en $O(m \alpha(m,n))$, où $\alpha(m,n)$ est l'inverse de la fonction d'Ackermann, qui est presque toujours inférieur à 5 en pratique (pour 2^{65535})

Algorithme incrémental de connexité

- On peut utiliser la structure de données de l'union-find pour calculer la connexité d'un graphe.
- Algorithme peut efficace, mais fonctionne si on connaît le graphe petit à petit
- Principes :
 - ▣ On fait makeSet(x) quand un nouveau sommet apparaît
 - ▣ On fait Union(x,y) quand une arête $\{x,y\}$ apparaît

Couplages

Exercices

262

- Donner un algorithme qui teste si un graphe est biparti
- Prouver cet algorithme

- Aide : on pourra faire une DFS ou une BFS

Couplage

263

- Un couplage dans un graphe est un ensemble d'arêtes qui n'ont pas de sommet en commun.
- C'est typiquement la solution aux problèmes d'appariements (regroupement par 2)
- Un **couplage maximum** est un couplage contenant le plus grand nombre possible d'arêtes.
- Un **couplage parfait** ou **1-factor** est un couplage M du graphe tel que tout sommet du graphe est incident à exactement une arête de M .

Couplage

264

- Soit M un couplage (matching en anglais),
- Un sommet qui est une extrémité d'une arête de M est dit **couplé** ou **"matché"**
- Un sommet qui n'est pas une extrémité de M est dit **libre**
- Une **chaîne alternée** est une chaîne dont les arêtes sont alternativement des arêtes du couplage et des arêtes n'appartenant pas au couplage
- Une **chaîne augmentante** est une chaîne alternée dont les deux extrémités sont libres

Chaîne augmentante

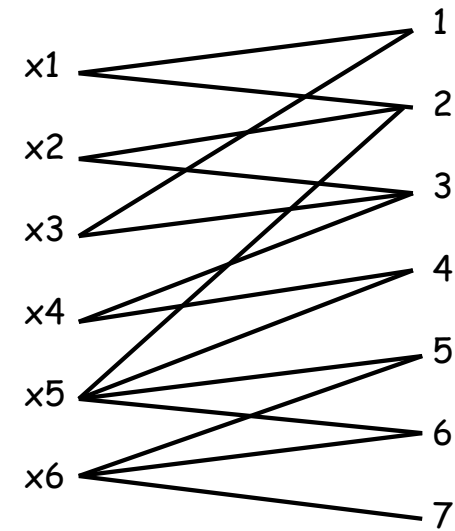
265

- Le terme vient du fait que si on trouve une chaîne augmentante on peut augmenter la cardinalité du couplage en inversant les arêtes de la chaîne : les libres deviennent matchées et les matchées deviennent libres.

Théorie des Graphes - 2015/2016

Couplage

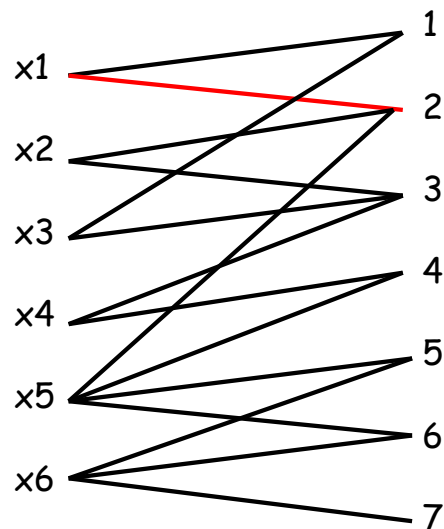
266



Théorie des Graphes - 2015/2016

Couplage

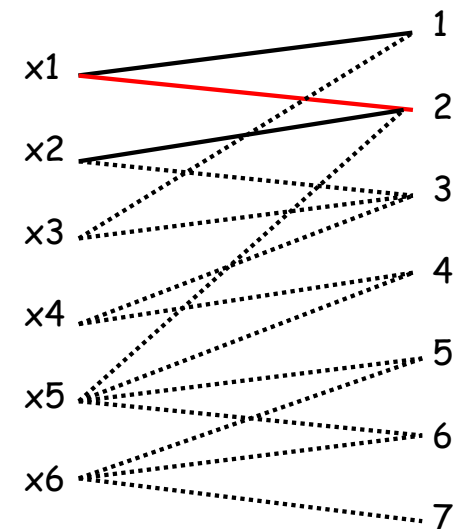
267



Théorie des Graphes - 2015/2016

Couplage

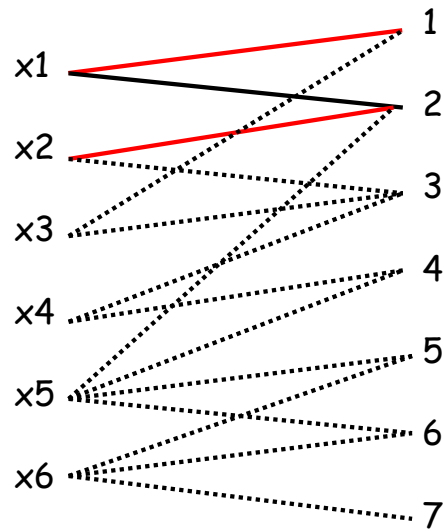
268



Théorie des Graphes - 2015/2016

Couplage

269



Théorie des Graphes - 2015/2016

Couplage : théorème de Berge

270

- Théorème : un couplage est maximum si et seulement si il n'admet pas de chaîne augmentante

Théorie des Graphes - 2015/2016

Couplage : algorithmes

271

- Le théorème nous fournit le principe d'un algorithme.
- On part d'un couplage vide
- On cherche une chaîne augmentante
 - ▣ Simple avec un graphe biparti
 - ▣ Complexe avec un graphe non biparti

Théorie des Graphes - 2015/2016

Couplage : graphe biparti

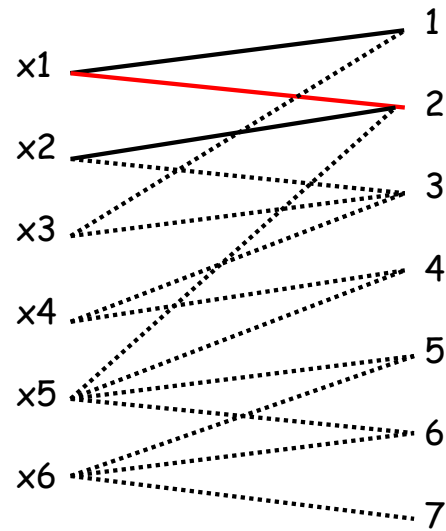
272

- Chercher une chaîne alternée dans un graphe biparti est simple, car l'alternance des arêtes correspond à l'alternance des ensembles.
 - ▣ Soit $G=(X \cup Y, E)$
 - ▣ On oriente les arêtes matchées de Y vers X, les autres de X vers Y.
 - ▣ Une chaîne augmentante devient un chemin d'un sommet libre de X vers un sommet libre de Y

Théorie des Graphes - 2015/2016

Couplage

273



Théorie des Graphes - 2015/2016

Couplage : graphe biparti

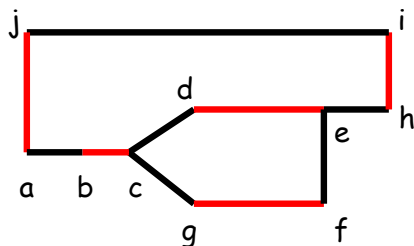
274

- Une DFS ou une BFS permet de trouver les chaînes augmentante
- La complexité est en $O(nm)$
- Si on travaille par niveau :
 - ▣ On sature tous les sommets atteignable au niveau i , avant de passer au niveau $i+1$
 - ▣ On peut montrer que la complexité est en $O(\sqrt{n} \cdot m)$

Théorie des Graphes - 2015/2016

Couplage : graphe non biparti

275

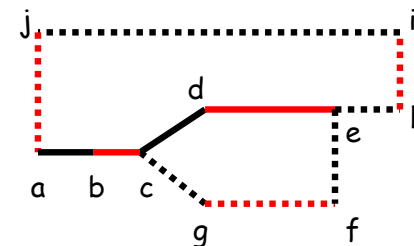


On ne peut plus orienté les arêtes comme avec les bipartis !

Théorie des Graphes - 2015/2016

Couplage : graphe non biparti

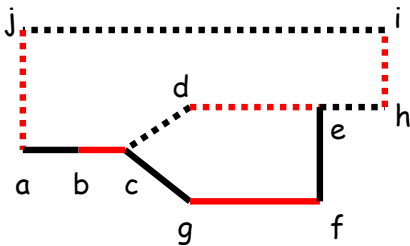
276



e est marqué pair

Théorie des Graphes - 2015/2016

Couplage : graphe non biparti



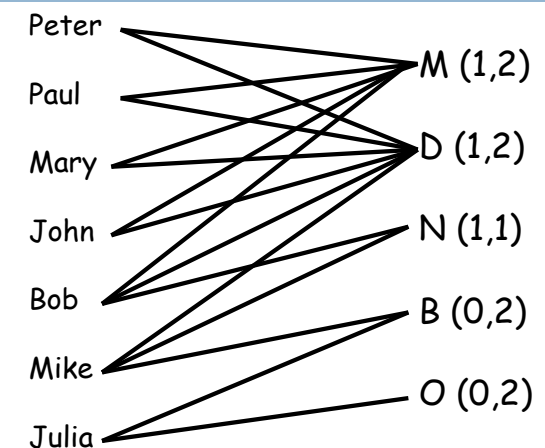
e est marqué impair

Couplage : graphe non biparti

- Idée: Edmond's algorithm
 - ▣ Shrinking-blossom algorithm
- Amélioration par Tarjan et d'autres
- Complexité $O(nm\alpha(n,m))$ difficile à implémenter
 $O(nm)$ très difficile
 $O(n^{1/2}m)$: 42 pages de démonstration non intuitive

Flots

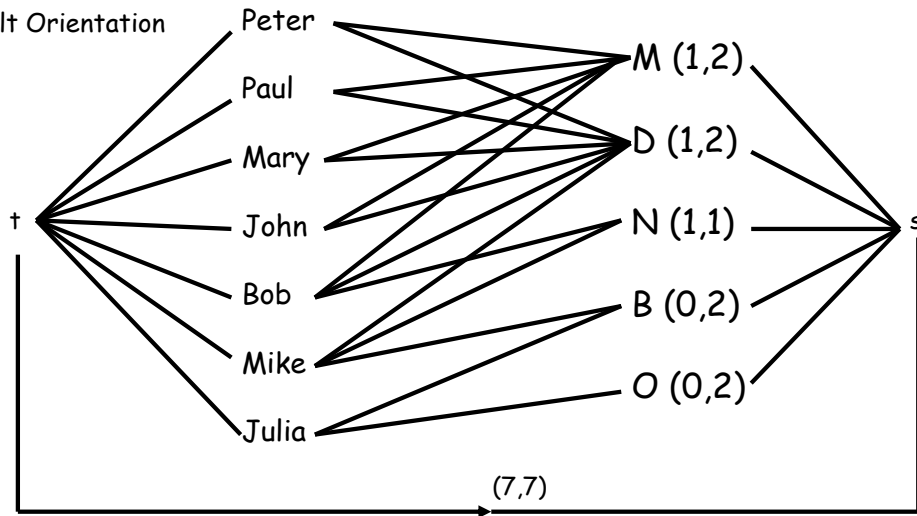
Flot : affectation



Affectation

281

Default Orientation

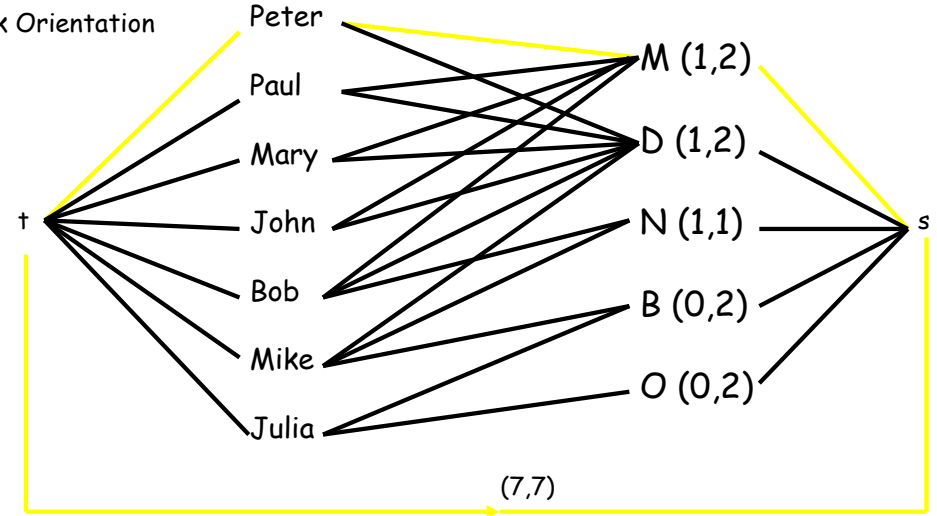


Théorie des Graphes - 2015/2016

Flot compatible

282

Black Orientation



Théorie des Graphes - 2015/2016

Augmentation successive

283

- Successive augmentation are computed in a particular graph:
The **residual graph**
- The residual graph has **no lower bounds**
- In our case this algorithm is equivalent to the best ones.

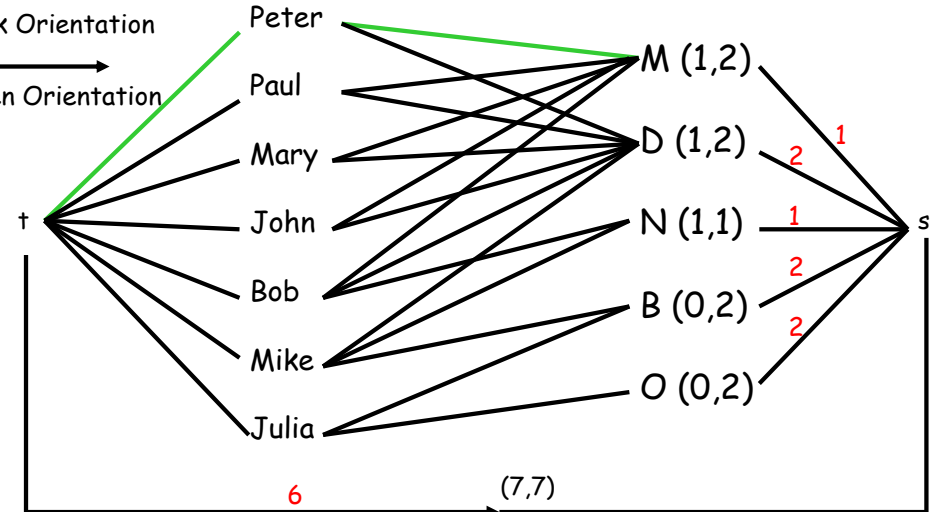
Théorie des Graphes - 2015/2016

Residual Graph

284

Black Orientation

Green Orientation

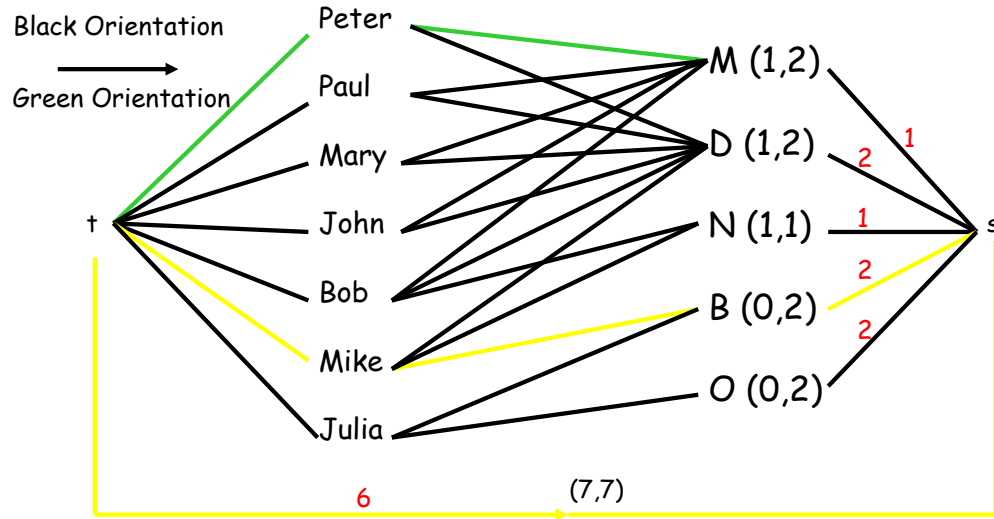


If $f(i,j) < u(i,j)$ then (i,j) and $r(i,j) = u(i,j) - f(i,j)$
If $f(i,j) > l(i,j)$ then (j,i) and $r(j,i) = f(i,j) - l(i,j)$

Théorie des Graphes - 2015/2016

Residual Graph

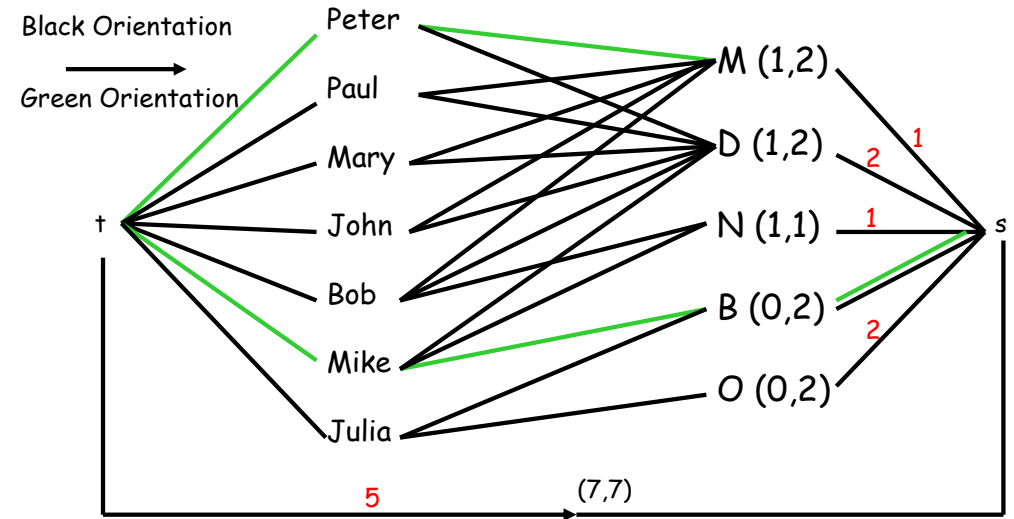
285



Théorie des Graphes - 2015/2016

Residual Graph

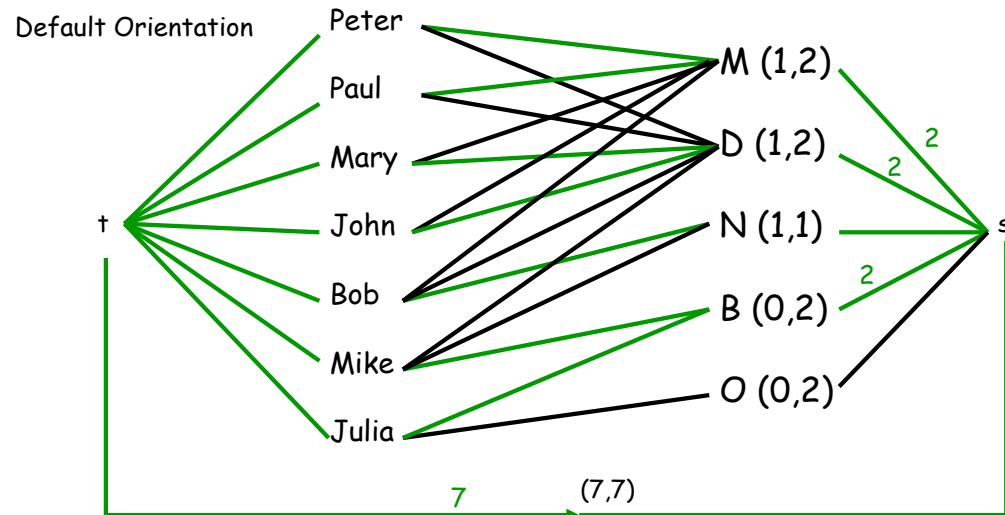
286



If $f(i,j) < u(i,j)$ then (i,j) and $r(i,j) = u(i,j) - f(i,j)$
If $f(i,j) > l(i,j)$ then (j,i) and $r(j,i) = f(i,j) - l(i,j)$

Une Solution

287



7 flow value

Théorie des Graphes - 2015/2016

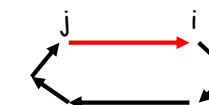
Properties

288

- The flow value x_{ij} of (i,j) can be increased iff there is a path from j to i in $R - \{(i,i)\}$



- The flow value x_{ij} of (i,j) can be decreased iff there is a path from i to j in $R - \{(i,j)\}$

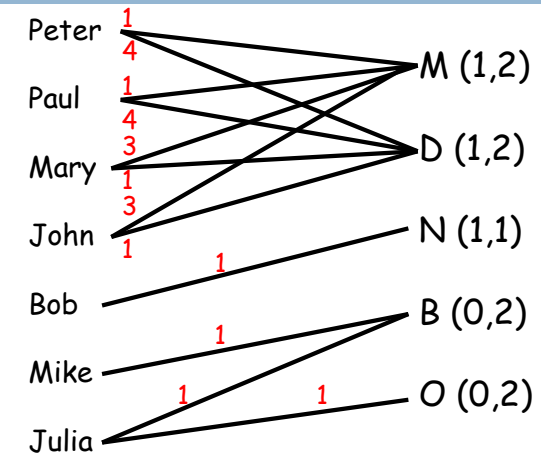


Théorie des Graphes - 2015/2016

Minimum cost Flows

- Let G be a graph in which every arc (i,j) is associated with 3 integers:
 - $l(i,j)$ the lower bound capacity of the arc
 - $u(i,j)$ the upper bound capacity of the arc
 - $c(i,j)$ the cost of carrying one unit of flow
- The cost of a flow f is $\text{cost}(f) = \sum f(i,j) c(i,j)$
- The minimum cost flow problem:
 - If there exists a feasible flow, find a feasible flow f such that $\text{cost}(f)$ is minimum.

Affectation avec coûts

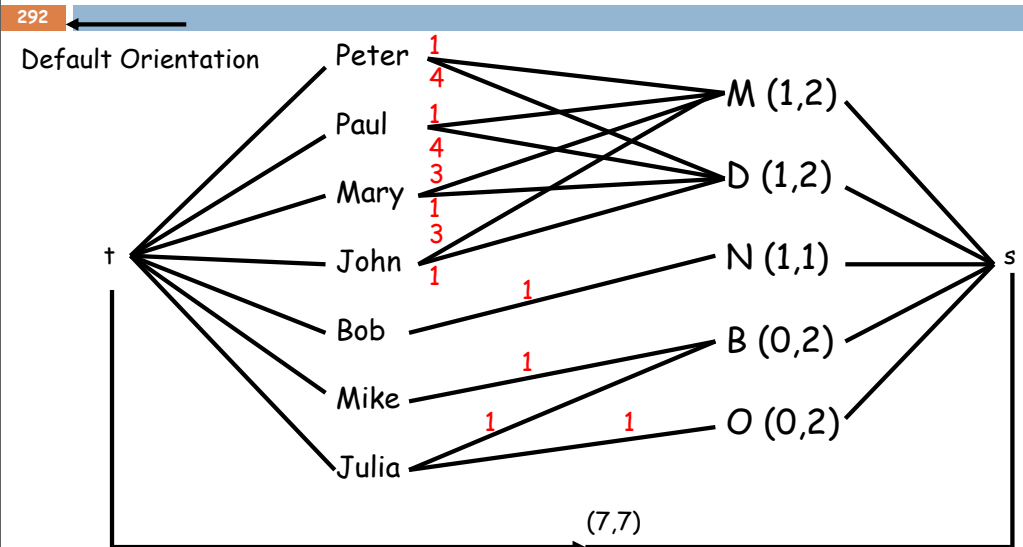


Sum < 12

Minimum Cost flow

- Similar to feasible flow except that: **shortest** paths are considered.
- length of an arc = **reduced cost** of the arc
- Reduced costs are used to work with nonnegative value (useful for shortest paths algorithms), but the principles remains the same with **residual costs**.
- We will consider here only the residual costs

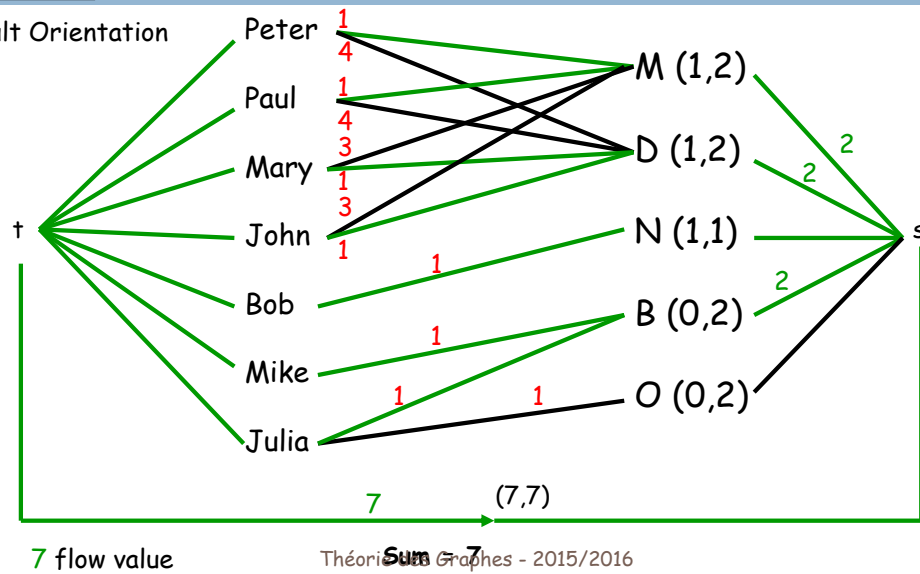
Minimum Cost Flow



Une Solution

293

Default Orientation

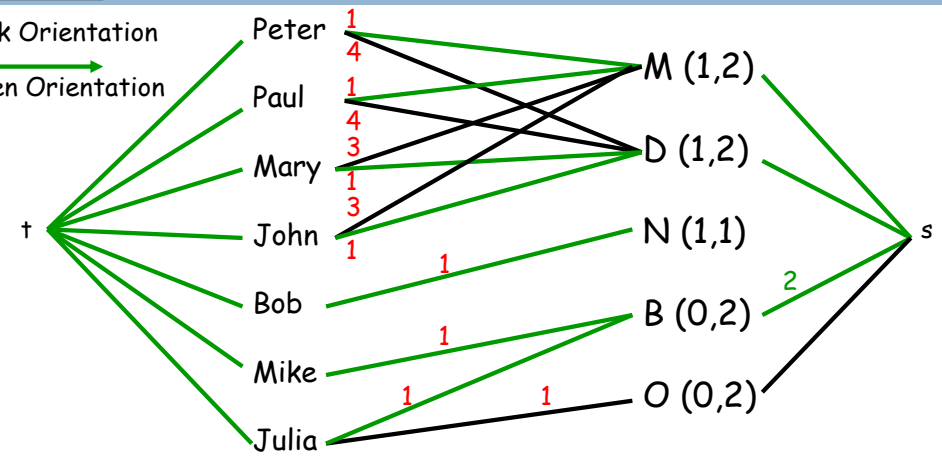


Residual Graph

294

Black Orientation

Green Orientation

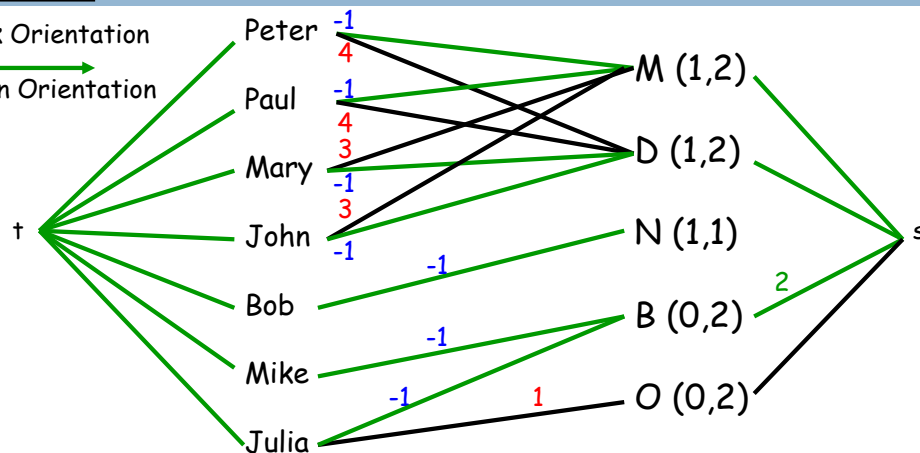


Residual Costs

295

Black Orientation

Green Orientation

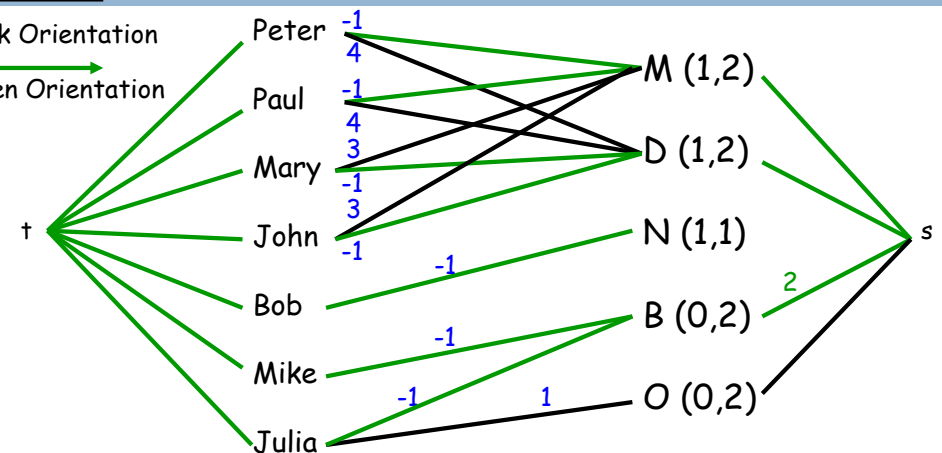


Residual Costs

296

Black Orientation

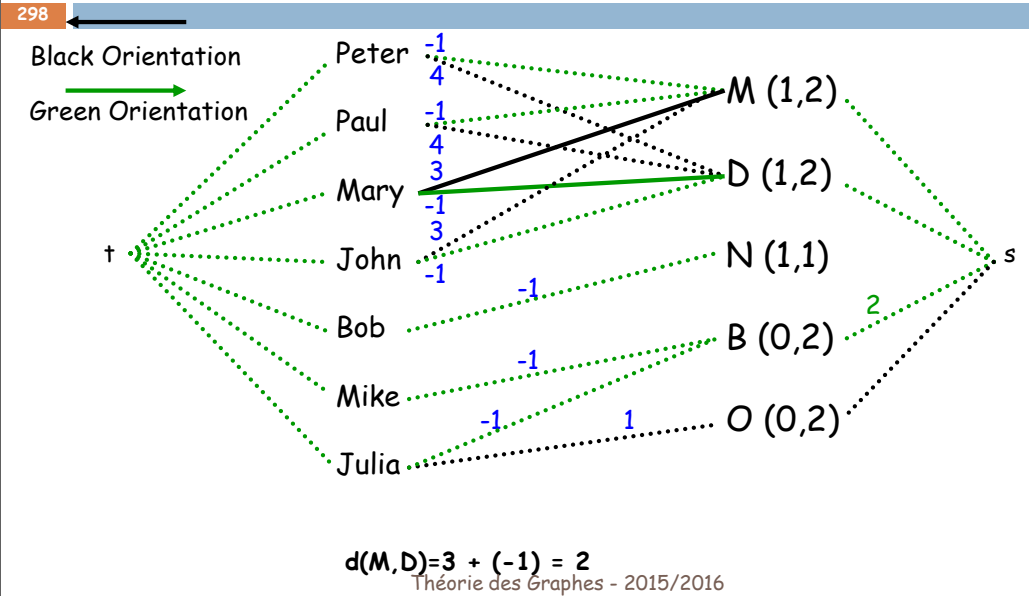
Green Orientation



Shortest path

- $d(i,j)$ = length of the shortest path which does not use (i,j) in the residual graph. The length is the sum of the residual costs of the arc contained in the path.

Residual Costs

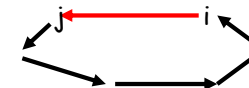


Minimum cost flow

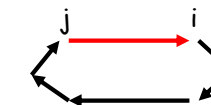
- If the feasible flow is computed by augmenting the flow along shortest paths then the solution is optimal.
- Complexity $O(n S(n,m,\chi))$ where χ is the maximum cost value.

Properties

- The flow value x_{ij} of (i,j) can be increased iff there is a path from j to i in $R - \{(i,j)\}$



- The flow value x_{ij} of (i,j) can be decreased iff there is a path from i to j in $R - \{(i,j)\}$



Terminaison de l'algorithme

301

- On peut éviter la dernière augmentation, en faisant attention à ce que le puits soit toujours atteignable dans le graphe résiduel.
- On utilise un compteur (num) qui compte le nb de nœuds que l'on peut atteindre à une distance k donnée depuis s . Si on change la distance d'un nœud de k_1 à k_2 alors on décrémente $\text{num}(k_1)$ et on incrémente $\text{num}(k_2)$. Si $\text{num}(k_1)$ devient vide alors on peut arrêter l'algorithme.
- Voir Ahuja/Magnanti et Orlin p 219

Théorie des Graphes - 2015/2016

Flots : push-relabel algorithm

302

Théorie des Graphes - 2015/2016

303

Théorie des Graphes - 2015/2016

Problèmes

304

- Trier les sommets d'un graphe en fonction de leur degré
 - ▣ Algorithme ?
 - ▣ Coût ?

Théorie des Graphes - 2015/2016

- Graphes planaires
- Plus courts chemins (Dijkstra, Bellman-Ford, les deux).
Recherche de distances entre deux points
- Cycles négatifs
- Couplage
- Flots
- Arbres recouvrants
- Ordonnancement, rang et tri topologique
- Graphes triangulés
- Graphes parfaits