

# Algorithmique des graphes

Licence 3, IUP IPL/ISI

2007-2008

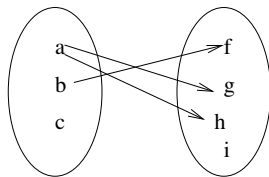
## Plan du cours

- Généralités sur les graphes et les relations binaires.
- Composantes connexes et fortement connexes. Graphes sans circuit. Partition en niveaux.
- Coloration de graphes. Nombre chromatique.
- Arbres et arborescences. Arbre partiel de coût minimum.
- Algorithmes de parcours et recherche de plus court chemin.
- Ordonnancement (méthode PERT)

## 1 Relations binaires et graphes

### Définition mathématique

Une relation binaire  $R$  consiste d'un ensemble  $A$  appelé domaine de  $R$ , d'un ensemble  $B$  appelé co-domaine de  $R$  et d'un sous-ensemble  $A \times B$  appelé le graphe de  $R$



$$A = \{a, b, c\} B = \{f, g, h, i\} G = \{(a, g), (a, h), (b, f)\}$$

### Propriétés (avec domaine=co-domaine)

La relation  $R$  sur  $A$  est

- réflexive :  $\forall x \in A xRx$
- symétrique :  $\forall x, y \in A$  si  $xRy$  alors  $yRx$
- transitive :  $\forall x, y, z \in A$ , si  $xRy$  et  $yRz$  alors  $xRz$

aussi : irréflexive, anti-symétrique

### Définition

Une relation d'équivalence est réflexive, symétrique et transitive.

- voisinage direct ?
- habiter dans la même rue ?
- habiter dans l'immeuble dans face ?
- avoir la même couleur de cheveux ?

### Partitions et relations d'équivalence

Une relation d'équivalence définit une partition de son domaine.

Chaque élément de cette partition est une *classe d'équivalence*

partition :  $P = \{X_1, X_2, \dots, X_n\}$  est une partition de  $E$  ssi

- $X_1 \cup X_2 \cup \dots \cup X_n = E$
- $\forall i, j X_i \cap X_j = \emptyset$

### Ordre partiel (strict)

propriétés :

- relation transitive :  $x < y$  et  $y < z \rightarrow x < z$
- relation antisymétrique :  $x < y$  implique non  $y < x$

### Ordre total

l'ordre est total si on peut comparer tous les éléments du domaine

soit  $x < y$  soit  $y < x$  quels que soient  $x$  et  $y$

exemple :  $<$  sur les entiers, les réels

contre-exemple : les nombres complexes ?

### Produit de relations

si on a deux relations  $R_1, R_2$  de domaines  $A_1, A_2$ , et de co-domaine  $B_1, B_2$ , on peut définir le produit des deux relations de  $A_1 \times A_2$  sur  $B_1 \times B_2$  (produit cartésien)

$$(a, b)R_{1 \times 2}(c, d) \text{ ssi } aR_1c \text{ et } bR_2d$$

### Graphe dirigé

$$G = (N, A)$$

$N$  est un ensemble de noeuds (ou sommets)

$A$  est un ensemble d'arcs,  $A \subseteq N \times N$

$(v, w) \in A$  est un arc de  $v$  à  $w$  (les extrémités de l'arc).

Un graphe est donc une relation binaire sur l'ensemble  $N$ .

### Quelques définitions

- L'**ordre** du graphe est son nombre de noeuds.
- Deux sommets sont dits **adjacents** s'ils sont reliés par une arête.
- Une **boucle** est une arête reliant un noeud à lui-même.

- Un **graphe simple** est le graphe d'une relation irreflexive (aucun noeud n'est relié à lui-même), symétrique.
- **Grappe non dirigé (non orienté)**  $(i, j) \in A \rightarrow (j, i) \in A$  c'est un graphe pour lequel l'orientation n'a pas d'importance (on représente tout lien par un lien simple).

### Quelques définitions

- On appelle **degré** d'un sommet le nombre d'arêtes dont ce sommet est une extrémité.
- un noeud sans arc sortant est un **puits**
- un noeud sans arc entrant est une **source**
- Si tous les sommets d'un graphe ont le même degré  $k$ , le graphe est dit **k-régulier**.
- Pour un graphe orienté, on définit le **degré entrant** ( $d^+$ ) et le **degré sortant** ( $d^-$ ) d'un noeud, pour le nombre d'arcs dont un noeud est l'origine ou la destination.
- Un graphe simple d'ordre  $n$  qui est  $(n-1)$ -régulier est dit **complet**. Deux sommets quelconques sont alors adjacents.

### Propriétés

- Pour un graphe non orienté  $(N, A)$  : Lemme des poignées de mains

$$\sum_{x \in N} d(x) = 2|A|$$

- Pour un graphe orienté  $(N, A)$  :

$$\sum_{x \in N} d^+(x) = \sum_{x \in N} d^-(x) = |A|$$

- Un graphe simple a au plus  $\frac{n(n-1)}{2}$  arêtes.

### Sous-graphe

soit un graphe  $G = (N, A)$ ,  $N' \subseteq N$ ,  $A' \subseteq A$ .  $G' = (N', A')$  est un sous-graphe de  $G$  si pour tout arête de  $A'$ , les extrémités de l'arête dans  $G$  sont dans  $N'$ .

### Propriété des relations

- relation irreflexive : le graphe ne comporte pas de boucle
- relation réflexive : ...
- relation transitive : pour tout chemin existant entre deux noeuds il existe un arc direct
- relation symétrique : pour tout arc reliant deux noeuds il y a le symétrique
- un graphe non dirigé est en fait le graphe d'une relation symétrique

### Graphes valués

On peut vouloir attribuer des valeurs aux arcs (distance, coût, etc) un graphe valué est alors donné par  $G = (N, A)$  où

$$A = \{(x_i, x_j, n_{ij}) / x_i \in N, x_j \in N, n_{ij} \in \mathbb{N}\}$$

### Matrice d'adjacence

- on numérote les noeuds
- $M[i,j]=1$  s'il y a un arc de  $i$  vers  $j$ , 0 sinon
- mémoire =  $O(N^2)$
- retrouver noeud/arc =  $O(1)$
- avantages du calcul matriciel

### Listes d'adjacence

- liste de  $(i, \{j_1, j_2, \dots\})$  où les  $j_k$  sont les successeurs de  $i$  dans le graphe.
- mémoire :  $O(N+A)$
- retrouver noeud  $O(N)$  (faisable en  $O(1)$ )
- retrouver arc  $O(N * \text{degré moyen}) = O(N * 2|A|/|N|) = O(A)$  ou tps de hachage (constant) ou bien  $|A|/|N|$

## 2 Connexité et parcours

### Chaînes et cycle (non orienté)

- chaîne une suite finie de sommets reliés entre eux par une arête (successivement)
- chaîne simple une chaîne qui n'utilise pas deux fois une même arête
- chaîne Eulérienne une chaîne simple qui passe par toutes les arêtes du graphe
- chaîne Hamiltonienne une chaîne simple qui passe par tous les sommets du graphe (une seule fois)
- chaîne élémentaire une chaîne qui ne passe pas deux fois par le même sommet (sauf éventuellement le départ et l'arrivée)
- cycle un cycle est une chaîne simple dont le départ est égal à l'arrivée.

### Chemins et circuits (orienté)

- **Chemin** Un chemin entre deux sommets  $v$  et  $w$  d'un graphe  $(N, A)$  est une suite  $C = (v, u_1, \dots, u_n, w)$  de noeuds de  $N$  tels que  $(v, u_1) \in A$ ,  $(u_i, u_{i+1}) \in A$  et  $(u_n, w) \in A$ . Si  $v = w$ , le chemin  $C$  est fermé. Si  $\forall i, j, u_i \neq u_j$ , le chemin est simple.
- Un **circuit** est un chemin fermé simple.
- Un **circuit eulérien** est un circuit simple passant par toutes les arcs une fois et une seule.
- Un **circuit hamiltonien** est un circuit simple passant par tous les sommets une fois et une seule.

un chemin est une chaîne dont tous les arcs sont orientés "dans le même sens".

### Définitions

- **Fermeture transitive** la fermeture transitive d'un graphe dirigé  $G = (N, A)$  est un graphe dirigé  $G^* = (N, A^*)$  tel que :  $(v, w) \in A^* \leftrightarrow$  il y a un chemin de  $v$  à  $w$  dans  $G$

## Définitions

- **Graphe non dirigé connexe**  $\forall i, j \in N, \exists$  une chaîne de  $i$  à  $j$

## Connexité

- **composante connexe** d'un graphe non dirigé : sous-graphe connecté maximal
- **composante fortement connexe** idem pour graphe orienté
- **graphe fortement connexe** graphe dirigé connexe : il existe un chemin pour toute paire de noeuds  $(u, v)$ ,  $u$  et  $v$  distincts (de  $u$  à  $v$  et de  $v$  à  $u$ ).

## Graphe réduit

Soit  $G = (N, A)$  un graphe orienté.

Le graphe réduit de  $G$ ,  $G_r = (N_r, A_r)$  est défini par :

- $N_r = \{n_1, n_2, \dots, n_p\}$ , les  $n_i$  étant les composantes fortement connexes de  $G$
  - $A_r = \{(n_i, n_j) / \exists x_i \in n_i \text{ et } x_j \in n_j \text{ tels que } (x_i, x_j) \in A\}$
- propriété : un graphe réduit est sans circuit

## Partition d'un graphe sans circuit

procédure pour trouver s'il y a un circuit : on trouve une partition en niveaux comme suit

1. on cherche les sommets sans arcs entrants (les sources)  
ils constituent le niveau 0,  $N_0$
2. on ignore ces noeuds et leurs arcs, et on cherche les noeuds qui sont alors des sources, pour faire le niveau  $N_1$
3. on continue jusqu'à ce qu'il n'y ait plus de noeuds (alors pas de circuits) ou plus de sources (alors il y a un circuit)

avec matrice d'adjacence : on cherche les lignes remplies de zéro, on enlève les lignes/colonnes correspondantes, on itère.

## Stabilité, ensembles stables de noeuds

- un **stable** de  $G$  est un sous-ensemble de sommets deux-à-deux non-adjacents

## Noyau d'un graphe

- Un noyau dans un graphe orienté est un ensemble  $N$  de sommets deux à deux non-adjacents tel que tout sommet hors de  $N$  a un successeur dans  $N$  (= stable et absorbant)

Tout graphe sans circuit a un noyau unique.

## Explorations

On appelle exploration ou parcours d'un graphe un procédé déterministe qui permet de choisir un sommet à partir de l'ensemble des sommets déjà visités de manière à passer par tous les sommets du graphe.

## Parcours en profondeur

Il consiste, à partir d'un noeud à toujours explorer un noeud voisin jusqu'à ce qu'il n'y en ait plus, puis à remonter dans le parcours pour visiter les voisins laissés de côté au fur et à mesure.

Pour cela il faut marquer les noeuds déjà visités. Un algorithme type est par exemple :

$G = (N, A)$  un graphe

$x$  : point de départ de l'exploration

$P$  : pile des sommets à traiter

$V$  : liste de sommets déjà traités

---

empiler( $P, x$ )

**while**  $P$  non vide **do**

$y \leftarrow$  depiler( $P$ )

$V \leftarrow V \cup \{y\}$

**for all**  $z$  tel que  $(y, z) \in A$  et  $z \notin P \cup V$  **do** ▷ nouveaux sommets non vus

    empiler( $P, z$ )

    pere[ $z$ ] ←  $y$  ▷ on conserve les chemins suivis

**end for**

**end while**

---

## Parcours en largeur

$G = (N, A)$  un graphe

$x$  : point de départ de l'exploration

$F$  : file des sommets à traiter

$V$  : liste de sommets déjà traités

---

ajouter( $F, x$ )

**while**  $F$  non vide **do**

$y \leftarrow$  defiler( $F$ )

$V \leftarrow V \cup \{y\}$

**for all**  $z$  tel que  $(y, z) \in A$  et  $z \notin F \cup V$  **do** ▷ nouveaux sommets non vus

    ajouter( $F, z$ )

    pere[ $z$ ] ←  $y$  ▷ on conserve les chemins suivis

**end for**

**end while**

---

### 3 Coloration de graphes

#### Mesures sur les graphes

- diamètre
- degré de connexité
- nombre chromatique
- nombre cyclomatique

#### Définition

- diamètre : longueur maximal du plus court chemin reliant deux noeuds quelconques d'un graphe
- degré de connexité : nombre minimal d'arêtes suffisant à rendre un graphe connexe non-connexe

#### Nombre chromatique

1. une **coloration** d'un graphe G est une fonction associant une valeur (une "couleur") à chaque noeud du graphe de telle sorte que deux noeuds reliés par un arc aient toujours une valeur différente
2. le nombre minimum de couleurs nécessaires pour colorier un graphe est appelé nombre chromatique, que l'on notera  $\gamma(G)$  ou  $\chi(G)$
3. cette notion n'a de sens que pour un graphe sans boucle.
4. une coloration effectue une partition du graphe en **stables**

#### Théorème des quatre couleurs

1. un graphe planaire est un graphe quelconque qui a la particularité de pouvoir se représenter sur un plan sans qu'aucune arête (ou arc pour un graphe orienté) n'en croise une autre.
2. Théorème : le nombre chromatique d'un graphe planaire est au plus 4.

#### Propriétés

- on peut encadrer ce nombre chromatique.
- Soit  $n$  l'ordre du graphe G
- soit  $\alpha(G)$  le cardinal de la plus grande partie stable
- soit  $r$  le degré maximal d'un noeud du graphe

$$\frac{n}{\alpha(G)} \leq \gamma(G) \leq r + 1$$

#### Propriétés

Soit G un graphe simple d'ordre  $n$  alors :

$$\gamma(G) + \alpha(G) \leq n + 1$$

#### Un algorithme de coloration approché : Welsh-Powell

$X \leftarrow$  liste des sommets, triés par ordre de degré décroissant  
 $C \leftarrow \emptyset$

```

for all  $s \in X$  do
  for all  $c \in C$  do
    if  $s$  n'est relié à aucun sommet colorié par  $c$  then
       $s$  est colorié en  $c$ 
    end if
  end for
  if  $s$  n'est pas colorié then
     $C \leftarrow C \cup \{c'\}$ 
    couleur de  $s \leftarrow c'$ 
  end if

```

▷ dans l'ordre de création

▷ on crée une nouvelle couleur

#### end for

version sans ordre sur les noeuds : algorithme "glouton"

#### Variante : D satur

idem Welsh et Powell

l'ordre des noeuds est selon

$$DS(s) = \begin{cases} d(s) & \text{si aucun voisin de } s \text{ n'est colorié} \\ \text{nb de couleurs utilisées dans le voisinage} & \text{sinon} \end{cases}$$

### 4 Arbres

#### Arbres

- Un arbre est un graphe connexe sans cycle.
- Un arbre de  $n$  noeuds a donc exactement  $n-1$  arêtes.
- Un arbre pointé est un arbre où on a distingué un sommet qq. Ce sommet est la racine de l'arbre.
- Pour un graphe orienté, la racine de l'arbre est un noeud qui n'a pas d'arcs entrants (il est unique).
- Une feuille est un noeud de degré 1 (graphe orienté : uniquement un arc entrant)

#### Arbre partiel/couvrant

- un arbre **couvrant** un graphe G est un arbre qui est aussi sous-graphe de G contenant tous les noeuds de G (= graphe partiel) (en anglais *spanning tree*).
- un graphe admet un arbre couvrant ssi il est connexe
- L'ajout d'un arc à l'arbre couvrant crée un cycle unique
- Tout arc supprimé à ce sous-graphe le rend sans cycle
- Tout couple de sommets est relié par une chaîne unique

### Arbre couvrant minimum

”Minimum Spanning Tree”

1. sur un graphe valué, avec valeurs positives. valeur d'un arc  $(u, v)$  noté  $w(u, v)$
2. graphe partiel
3. somme des valeurs des arêtes minimale

$$w(T) = \sum_{(u,v) \in T} w(u, v)$$

nécessairement sans cycle sinon on pourrait enlever une arête.

### Algorithme de Kruskal

trier les arêtes en ordre de poids croissant et les ranger dans une liste L.

```

nb sommets ← 0
T ← ∅
poids ← 0
while nb sommets < n - 1 do
  e ← première arête de L
  L ← L / {e}
  if T ∪ {e} ne contient pas de cycle then
    T ← T ∪ {e}
    nb sommets ← nb sommets + 1
    poids ← poids + p(e)
  end if
end while

```

### Algorithme de Prim : résumé

Soit  $G=(V,E)$  un graphe

Soit A les noeuds de l'arbre en construction, T l'ensemble de ses arêtes.

```

T = ∅
A = {x}
while |T| < |V| - 1 do
  e ← (x, y) tq x ∈ A, y ∉ A et (x,y) de poids minimum
  T ← T ∪ {e}
  A ← A ∪ {y}
end while

```

▷ au hasard

### Algorithme de Prim

On garde les noeuds non encore traités dans une file à priorité Q, les noeuds étant indexés avec le poids de l'arc minimum qui le connecte à un sommet de A

```

Q ← V
index[v] ← ∞ pour tout v ∈ V
index[x] ← 0 pour un x choisi arbitrairement (pt de départ)
while Q ≠ ∅ do
  u ← extract-min(Q)
  for all v adjacent à u do
    if v ∈ Q et w(u, v) < index[v] then
      index[v] ← w(u, v) (mise à jour de la file)
      pere[v] ← u
    end if
  end for
end while

```

L'arbre couvrant minimal est alors donné par  $\{(v, pere[v])\}$

## 5 Algorithmes de plus courts chemins

### 5.1 Valeurs positives uniquement

#### Algorithme de Dijkstra

$G=(N,A)$  : graphe valué positivement ; valeur de l'arc  $(i,j)$  noté  $val(i,j)$

$i_0$  : point de départ

$\lambda_i$  : distance trouvée la plus courte de  $i_0$  au sommet  $i$

V : sommets déjà traités

```

λi0 ← 0
for all i ∈ N, i ≠ i0 do λi ← ∞
end for
S ← ∅
while S ≠ N do
  choisir i ∈ N/S tel que λi = minj ∈ N/S(λj)
  // si on cherche seulement le chemin le plus court vers ce i particulier,
  // on peut s'arrêter là
  S ← S ∪ {i}
  for all j ∈ Γ+(i) ∩ (N/S) do
    if λj > λi + val(i, j) then
      λj ← λi + val(i, j)
      pere[j] ← i
    end if
  end for
end while

```

▷ on choisit le sommet non exploré le plus proche du départ

▷ parmi les suivants de i non explorés

▷ a-t-on trouvé un chemin plus court vers j ?

▷ alors on le mémorise

## 5.2 Valeurs quelconques

### Algorithme de Moore

$G=(N,A)$  : graphe valué ; valeur de l'arc  $(i,j)$  noté  $w(i,j)$   
 $i_0$  : point de départ  
 $\lambda_i$  : distance trouvée la plus courte de  $i_0$  au sommet  $i$   
 $S$  : sommets à traiter

---

```

 $\lambda_{i_0} \leftarrow 0$ 
for all  $i \in N, i \neq i_0$  do  $\lambda_i \leftarrow \infty, v_i = (0,0)$ 
end for
 $S \leftarrow \{i_0\}$ 
while  $S \neq \emptyset$  do
    choisir  $i \in S$  tel que  $\lambda_i = \min_{j \in S}(\lambda_j)$             $\triangleright$  on choisit le sommet à traiter le plus
                                                            $\triangleright$  "proche" du départ
     $S \leftarrow S \setminus \{i\}$ 
    for all  $j \in \Gamma^+(i)$  do
        if  $\lambda_j > \lambda_i + w(i,j)$  then            $\triangleright$  a-t-on trouvé un chemin plus court vers j ?
            if l'ajout de  $(i,j)$  aux arcs retenus  $(\bigcup_{k \in N/v_k \neq (0,0)} vk)$  crée un cycle then
                retourner "cycle négatif trouvé"
            else
                 $\lambda_j \leftarrow \lambda_i + w(i,j)$             $\triangleright$  alors on le mémorise
                 $v_j \leftarrow (i,j)$             $\triangleright$  on mémorise le père de  $j$ 
                 $S \leftarrow S \cup \{j\}$ 
            end if
        end if
    end for
end while

```

---

Avec cet algorithme, on peut remettre dans  $S$  un sommet déjà rencontré s'il est amélioré ; s'il crée un cycle négatif, on le détecte.

### Algorithme de Bellman-Ford

Itération sur les arcs pour calculer les plus courts chemins en partant d'un point  $i_0$   
 $G=(N,A)$  : graphe valué ; valeur de l'arc  $(i,j)$  noté  $w(i,j)$   
 $i_0$  : point de départ  
 $\lambda_i$  : distance trouvée la plus courte de  $i_0$  au sommet  $i$

---

```

 $\lambda_{i_0} \leftarrow 0$ 
for all  $u \in N, u \neq i_0$  do  $\lambda_u \leftarrow \infty$ 
end for
for  $i = 1$  à  $|N| - 1$  do            $\triangleright$   $i$  compte juste le nb d'itérations
    for all  $(u,v) \in A$  do
        if  $\lambda_v > \lambda_u + w(u,v)$  then            $\triangleright$  mise à jour
             $\lambda_v \leftarrow \lambda_u + w(u,v)$ 
             $pere[v] \leftarrow u$ 
        end if
    end for
end for
for all  $(u,v) \in A$  do            $\triangleright$  vérification qu'on est arrivé à stabilité
    if  $\lambda_v > \lambda_u + w(u,v)$  then            $\triangleright$  il reste encore un chemin à améliorer
        cycle négatif détecté
    end if
end for

```

## 6 Ordonnement

### Chemin le plus long dans graphe acyclique et dates au plus tôt

fin : durée minimale du projet =  $t_{x_{\text{fin}}}$

---

$N_1, N_2, \dots, N_p$  = découpage en niveau du graphe

$t_{x_0} \leftarrow 0$

**for all**  $k=1 \dots p$  **do**

**for all**  $x_j \in N_k$  **do**

$t_{x_j} = \max\{w(x_i, x_j) + t_{x_i} \mid x_i \in N_{z < k} \text{ et } (x_i, x_j) \in A\}$

**end for**

**end for**

Pour avoir les dates au plus tard des étapes intermédiaires (date au plus tard à laquelle on peut démarrer sans retarder le projet), on fait la même chose en partant de  $N_p$  et en remontant vers  $N_0 = \{x_0\}$ , et en prenant le min au lieu du max dans l'algorithme ci-dessus.

Le plus long chemin de  $x_0$  à  $x_{\text{fin}}$  est appelé *chemin critique* : les sommets-tâches de ce chemin sont les *tâches critiques*. Pour ces tâches, la date au plus tard est égale à la date au plus tôt.

Tout retard dans l'exécution de l'une quelconque de ces tâches allonge d'autant la durée minimale du projet.

## 7 Problèmes de flots

### Définition d'un flot

Un *flot* sur un graphe est un vecteur  $\varphi = (\varphi^1, \dots, \varphi^m)$  de  $\mathbb{R}^m$  tel que le nombre  $\varphi_i = \varphi(u_i)$  est appelé *flux* dans l'arc  $u_i$  et qui vérifie la *loi de Kirchhoff* ou *loi de conservation du flux* en chaque sommet du graphe :

$$\forall x \in X, \quad \sum_{u_i \in \omega^-(\{x\})} \varphi(u_i) = \sum_{u_i \in \omega^+(\{x\})} \varphi(u_i)$$

Le flux  $\varphi_i$  dans l'arc  $u_i$  peut être assimilé à la quantité de véhicules (de liquide, d'information) parcourant l'arc  $u_i$  dans le sens de son orientation si  $\varphi_i > 0$  ou dans le sens inverse si  $\varphi_i < 0$ . La loi de Kirchhoff impose qu'en tout sommet le flux entrant soit égal au flux sortant.

$\omega^-(\{x\})$  (respectivement  $\omega^+(\{x\})$ ) est l'ensemble des arcs entrant dans (respectivement sortant de)  $x$ .

### Réseau de transport

Un *réseau de transport* est un graphe orienté connexe  $R = (X, U = \{u_0, \dots, u_m\})$  avec

– un sommet sans prédecesseur appelé *entrée* (ou source) noté  $s$  ( $\Gamma^-(s) = \emptyset$ )

– et un sommet sans suivant appelé *sortie* (ou puits) noté  $t$  ( $\Gamma^+(t) = \emptyset$ )

– et une application  $c : U \rightarrow \mathbb{R}^+ \cup \{\infty\}$  qui à chaque arc  $u$  associe sa capacité  $c(u) \geq 0$ .

L'arc  $u_0$  est un arc fictif  $u_0 = (t, s)$  de capacité infinie qui sera appelé *arc de retour*.