Algorithmes de Graphe

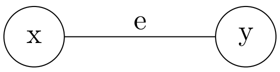
Pierre E. Meunier, Loïc Magnan

14 mars 2007

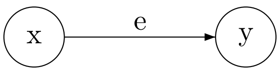
1 Définitions de base

1.1 Graphes non orientés / orientés

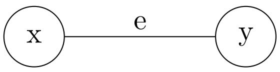
Soit un graphe $G = (V, E)$ non orienté, $E \subseteq P_2(V)$, et une arête $e = xy \in E$. On représente G par :



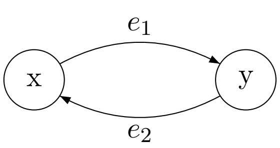
$G = (V, E)$ un graphe orienté, $E \subseteq V \times V$, et une arête $e = (x, y) \in E$



On peut représenter un graphe non orienté par un graphe orienté :

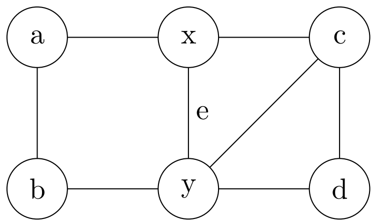


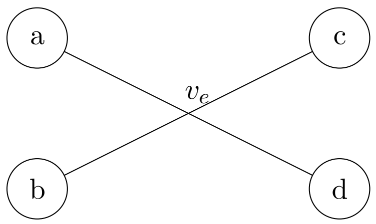
est équivalent à :



1.2 Définitions

- Boucles, multi-arêtes (multi-arcs)
- Adjacence : on dit que x et y sont adjacents si $xy \in E$
- Incidence : On dit que e est incidente à x et y si $e = xy$
- Voisinage : Soit $x \in V$. Il y a deux cas :
  - Non orienté : $N(x) = y \in V | yx \in E$ est un voisinage ouvert de x, et $N[x] = y \in V | yx \in E$ ou $y = x$ est un voisinage fermé de x
  - Orienté : $N^+(x) = y \in V | (x, y) \in E$ est un voisinage sortant de x, et $N^-(x) = y \in V | (y, x) \in E$ est un voisinage entrant de x
- Degrés : soit $x \in V$, on a : $deg(x) = |N(x)|$ le degré de x, $deg^+(x) = |N^+(x)|$ le degré sortant de x, et $deg^-(x) = |N^-(x)|$ son degré entrant.
- Sous-structures : Soit $G = (V, E)$ un graphe non orienté. Un *sous-graphe* de G est un graphe induit par $X \subseteq V$ noté $G[X] = (X, E \cap X \times X)$ (on enlève les sommets pour ne garder que ceux de X). Un *graphe partiel* est un graphe construit en enlevant des arêtes de G : pour $F \subseteq E$, $G_F = (V, F)$. On peut construire des sous-graphes partiels.
- Mineur : graphe obtenu en supprimant des arêtes et sommets, et en contractant des arêtes :





- Clique de G : Soit $X \subseteq V$ tel que $\forall x, y \in X, xy \in E$
- Indépendant de G : $X \subseteq V$ tel que $\forall x, y \in X, xy \notin E$ (graphes non-orientés)
- Parcours : $\mu = (x_1, x_2, ..., x_k)$ tel que $x_i x_{i+1} \in E$
- Chaîne (chemin) : parcours qui ne repasse pas plusieurs fois par la même arête
- Cycle : chaîne fermée $x_1 = x_k$ (circuit)
- Chemin élémentaire : ne passe pas plusieurs fois par le même sommet
- Longueur d'une chaine, d'un chemin, d'un cycle, d'un circuit : nombre d'arêtes (arcs)
- Distance de x à y : longueur minimal d'une chaîne de x à y s'il existe ($-\infty$ sinon)
- Connexité (graphes non-orientés) : $\forall x, y \in V, \exists$ une chaîne de x à y
- Forte connexité (graphes orientés) : $\forall x, y \in V, \exists$ une chaîne de x a y et $\exists$ une chaine de y à x

– Diamètre (graphes non-orientés) : $max_{x,y \in V} dist(x, y)$

1.3 Premières propriétés

Théorème 1.3.1. *Soit $G = (V, E)$ un graphe non orienté, $n = |V|$, $m = |E|$. On a :*

$$\sum_{x \in V} deg(x) = 2n$$

Démonstration. Dans la somme, chaque arête est comptée exactement deux fois (une fois pour chaque extrémité) □

Théorème 1.3.2. *Soit $G = (V, E)$ un graphe orienté. Alors*

$$\sum_{x \in V} deg^+(x) = m$$

Utilisation mathématique : parité.
Corollaire : le nombre de sommets de degré impair est pair.
Algorithme : complexités en $O(\sum_{x \in V} deg(x)) = O(m)$

2 Parcours

2.1 Représentation d'un graphe (Structure de données)

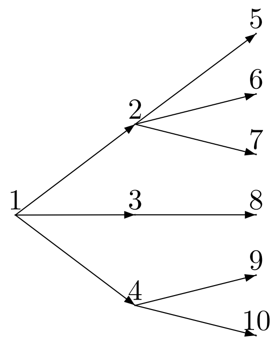
- Les classiques : $G = (V, E)$ orienté ou pas.
  - Matrices d'adjacence : M de taille $n \times m$. $M[x, y] = 1$ si $(x, y) \in E$ ($xy \in E$) et $M[x, y] = 0$ sinon
  - Listes d'adjacence : A tableau où, pour $x \in V$, $A[x]$ est la liste des éléments de $N^+(x)$ (ou $N(x)$) (listes chainées ou doublement chainées). Rien n'interdit d'utiliser d'autres structures de données pour stocker $N(x)$ ou $N^+(x)$.
- Autres cas de structures de données. Exemple : on peut représenter les sommets par des intervalles de R (réels), et les arêtes sont représentées par des relations particulières entre sommets, par exemple $II' \in E$ si $I \cap I' \neq \emptyset$
- Compromis espace / temps :

	matrice d'adjacence	liste d'adjacence	intervalles d'un graphe d'intervalles
espace	$\Theta(n^2)$	$\Theta(n + m)$	$\Theta(n)$
test d'adjacence entre x et y	$O(1)$	$O(n)$	$O(1)$
récupérer $N(x)$	$\Theta(n)$	$\Theta(deg(x))$	$\Theta(n)$

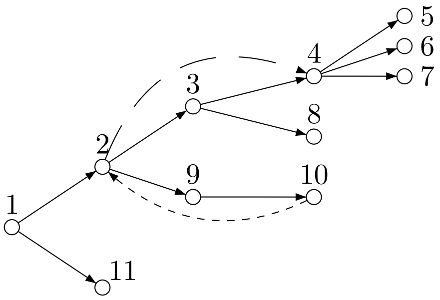
- Algorithme générique :
  à_traiter $\leftarrow \emptyset$
  $\forall x \in V$, état$[x] \leftarrow$ pas découvert
  $\forall x \in V$, faire

si état$[x]$ = pas découvert alors
état$[x] \leftarrow$ atteint
état$[x] \leftarrow$ NULL (x est racine)
à_traiter $\leftarrow$ à_traiter$\cup x$
tant que à_traiter $\neq \emptyset$ faire
$y \leftarrow$ choisir dans à_traiter
à_traiter $\leftarrow$ à_traiter $-y$
$\forall z \in N^+(y)$
faire si etat$[z]$ = pas découvert alors
père$[z] \leftarrow y$
etat$[z] \leftarrow$ atteint
à_traiter $\leftarrow$ à_traiter $\cup z$
etat$[y] \leftarrow$ traité

- Variantes de parcours suivant :
  - SD pour `à_traiter`
  - choisir `dans à_traiter`
- Parcours en largeur (BFS : Breadth First Search) :
  On représente a_traiter par une structure FIFO (file). On a :
  choisir : lire tête
  \ : enlever tête
  $\cup$: mettre en queue



  BFS permet en outre de calculer les distances (en nombre d'arêtes ou arcs) depuis le sommet de départ.
- Parcours en profondeur (DFS : Depth First Search)
  à_traiter : LIFO (pile)
  choisir : lire tête
  \ : dépiler
  $\cup$: empiler



- Complexité :

	Matrice d'adjacence	Listes d'adjacence
BFS	$\Theta(n^2)$	$\Theta(n+m)$
DFS	$\Theta(n^2)$	$\Theta(n+m)$

En effet, chaque sommet passe une fois et une seule dans `à_traiter`. Cela justifie la complexité en $\Theta(n^2)$. Dans le cas orienté, les arcs sont vus une fois, dans le cas non orienté, les arêtes sont vues deux fois.