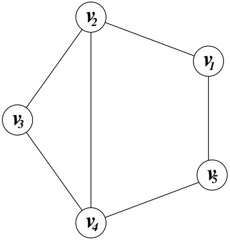
Cours 6
Algorithmes de graphes

L2 Informatique

Université de Montpellier

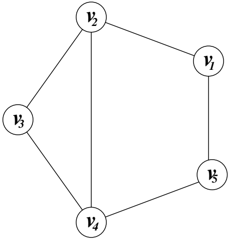
1- Graphes (1/3)

- Un graphe (fini) $G = (V, E)$ est constitué :
  - d'un ensemble (fini) de sommets V (ou V(G)) de taille n
  - d'un ensemble d'arêtes E (ou E(G)), paires d'éléments de V, de taille m.



1- Graphes (1/3)

- Un graphe (fini) $G = (V, E)$ est constitué :
  - d'un ensemble (fini) de sommets V (ou V(G)) de taille n
  - d'un ensemble d'arêtes E (ou E(G)), paires d'éléments de V, de taille m.



- Deux sommets $x, y \in V$ tels que $\{x, y\} \in E$ sont dits voisins, reliés ou adjacents.
  On note $\{x, y\} \in E$ ou $\mathrm{xy} \in \mathrm{E}$ (ou $yx \in E$). L'arête xy est incidente aux sommets x et y qui sont ses extrémités.
  Si deux arêtes ont une extrémité en commun, elles sont adjacentes, sinon elles sont disjointes.

1- Graphes (2/3)

- Les graphes considérés dans ce cours ne contiennent ni boucle (arête de type xx) ni d'arête multiple (arête en plusieurs exemplaires).

- Bestaire : Chemin, cycle, couplage, graphe complet, graphe vide, graphe biparti complet

1- Graphes (2/3)

- Les graphes considérés dans ce cours ne contiennent ni boucle (arête de type xx) ni d'arête multiple (arête en plusieurs exemplaires).
- Bestaire : Chemin, cycle, couplage, graphe complet, graphe vide, graphe biparti complet

Lemme (Nbre max d'arêtes)

Tout graphe G vérifie $m \leq \frac{n(n-1)}{2}$.

1- Graphes (3/3)

- Le voisinage de x, noté $\mathrm{N}_G(\mathrm{x})$, est l'ensemble des voisins du sommet x.

- Le degré d'un sommet x est le nombre de ses voisins, on le note $d_G(x)$ (autrement dit $d_G(x) = |N_G(x)|$).

1- Graphes (3/3)

- Le voisinage de x, noté $\mathrm{N}_G(\mathrm{x})$, est l'ensemble des voisins du sommet x.
- Le degré d'un sommet x est le nombre de ses voisins, on le note $d_G(x)$ (autrement dit $d_G(x) = |N_G(x)|$).

Théorème (Formule des degrés)

$$\sum_{x \in V(G)} d_G(x) = 2m$$

- Un graphe est k-régulier si les degrés de tous ses sommets valent k.

2- Codage

- Généralement, V est codé par $\{1, \ldots, n\}$ ou $\{0, \ldots, n-1\}$
- E peut classiquement être encodé par :
  - Liste d'arêtes, de taille $O(m)$, le test d'existence se faisant en $O(m)$.
  - Liste de voisins (pour chaque sommet v, on stocke la liste $L(v)$ de ses voisins), de taille $O(m)$, le test d'existence se faisant en $O(m)$.
  - Matrice d'adjacence A où $A_{i,j} = 1$ si les sommets i et j sont adjacents, 0 sinon, de taille $O(n^2)$, le test d'adjacence se faisant en $O(1)$.

3- Sous-graphes

- Deux graphes G et G' sont isomorphes si il existe une bijection f de $V(G)$ dans $V(G')$ telle que pour tout $x, y \in V(G)$ on ait $xy \in E(G) \;\Leftrightarrow\; f(x)f(y) \in E(G')$. La fonction f est un isomorphisme entre G et G'. On considèrera (improprement...) que G et G' sont égaux si il existe un isomorphisme entre eux.

3- Sous-graphes

- Deux graphes G et G' sont isomorphes si il existe une bijection f de $V(G)$ dans $V(G')$ telle que pour tout $x, y \in V(G)$ on ait $xy \in E(G) \Leftrightarrow f(x)f(y) \in E(G')$. La fonction f est un isomorphisme entre G et G'. On considèrera (improprement...) que G et G' sont égaux si il existe un isomorphisme entre eux.
- Soient G et H deux graphes.
  - Si $V(H) \subseteq V(G)$ et $E(H) \subseteq E(G)$ alors H est un sous-graphe de G.
  - Si $V(H) = V(G)$ et $E(H) \subseteq E(G)$ alors H est un sous-graphe couvrant de G.
  - Si $V(H) \subseteq V(G)$ et $E(H) = \{uv \ : \ uv \in EG(G),\ u \in V(H),\ v \in V(H)\}$ alors H est un sous-graphe induit de G.

On dira (improprement...) que G contient H si H est isomorphe à un sous-graphe de G.

3- Sous-graphes

- Deux graphes G et G' sont isomorphes si il existe une bijection f de $V(G)$ dans $V(G')$ telle que pour tout $x, y \in V(G)$ on ait $xy \in E(G) \;\Leftrightarrow\; f(x)f(y) \in E(G')$. La fonction f est un isomorphisme entre G et G'. On considèrera (improprement...) que G et G' sont égaux si il existe un isomorphisme entre eux.
- Soient G et H deux graphes.
  - Si $V(H) \subseteq V(G)$ et $E(H) \subseteq E(G)$ alors H est un sous-graphe de G.
  - Si $V(H) = V(G)$ et $E(H) \subseteq E(G)$ alors H est un sous-graphe couvrant de G.
  - Si $V(H) \subseteq V(G)$ et $E(H) = \{uv \;:\; uv \in EG(G),\; u \in V(H),\; v \in V(H)\}$ alors H est un sous-graphe induit de G.

  On dira (improprement...) que G contient H si H est isomorphe à un sous-graphe de G.
- Pour $X \subseteq V(G)$ on note G[X] le sous-graphe induit de G par les sommets de X. On note G \ X le graphe $G[V(G) \setminus X]$.

4- Connexité, arbres (1/2)

- Un chemin de G d'extrémités x et y est appelé un xy-chemin.
- Un graphe G est connexe si pour tous sommets x et y de G, le graphe G contient un xy-chemin.

4- Connexité, arbres (1/2)

- Un chemin de G d'extrémités x et y est appelé un xy-chemin.

- Un graphe G est connexe si pour tous sommets x et y de G, le graphe G contient un xy-chemin.

  Une composante connexe de G est un ensemble de sommets de G qui induit un sous-graphe connexe de G et qui est maximal pour cela. Si G est connexe alors il possède une seule composante connexe.

- Un arbre est un graphe connexe et sans cycle. Une forêt est un graphe sans cycle. Une feuille est un sommet ayant exactement un voisin.

4- Connexité, arbres (2/2)

Théorème (Propriétés des arbres et forêts)

Un arbre ayant au moins deux sommets contient au moins deux feuilles. Une forêt ayant c composantes connexes possède $n - c$ arêtes.

4- Connexité, arbres (2/2)

Théorème (Propriétés des arbres et forêts)

Un arbre ayant au moins deux sommets contient au moins deux feuilles. Une forêt ayant c composantes connexes possède $n - c$ arêtes.

Théorème (Arbres couvrants)

Un graphe G est connexe ssi il possède un arbre couvrant.

4- Connexité, arbres (2/2)

Théorème (Propriétés des arbres et forêts)

Un arbre ayant au moins deux sommets contient au moins deux feuilles. Une forêt ayant c composantes connexes possède $n - c$ arêtes.

Théorème (Arbres couvrants)

Un graphe G est connexe ssi il possède un arbre couvrant.

- Un chemin de longueur minimum entre deux sommets x et y est appelé un plus court chemin de x à y et sa longueur est la distance de x et y, notée $\mathrm{dist}_G(x,y)$.

5- Parcours

- Soit $G = (V, E)$ un graphe connexe, un parcours de G est donné par :
  - une énumération $v_1, \ldots, v_n$ des sommets de V
  - une fonction $pere : V \rightarrow V$ telle que $pere(v_1) = v_1$ et pour $i \geq 2$: $pere(v_i) \in \{v_1, \ldots, v_{i-1}\}$ et $v_i pere(v_i)$ est une arête de G.

5- Parcours

- Soit $G = (V, E)$ un graphe connexe, un parcours de G est donné par :
  - une énumération $v_1, \ldots, v_n$ des sommets de V
  - une fonction $pere : V \to V$ telle que $pere(v_1) = v_1$ et pour $i \geq 2$: $pere(v_i) \in \{v_1, \ldots, v_{i-1}\}$ et $v_i pere(v_i)$ est une arête de G.

- Le sommet v_1 est appelé la racine du parcours et $\{v_i pere(v_i) \ : \ i \geq 2\}$ forme les arêtes du parcours.

Théorème (Parcours de graphes)

Pour tout graphe connexe G et tout sommet r de G, le graphe G admet un parcours de racine r et les arêtes de tout parcours de G forment un arbre couvrant de G.

6- Parcours en largeur (1/2)

- Pour un graphe connexe $G = (V, E)$ et r un sommet de G, un arbre des plus courts chemins depuis r est un arbre T couvrant G et vérifiant : pour tout sommet x de G on a $dist_T(r, x) = dist_G(r, x)$. On note $n_T(x)$ la valeur $dist_T(r, x)$.

6- Parcours en largeur (1/2)

- Pour un graphe connexe $G = (V, E)$ et r un sommet de G, un arbre des plus courts chemins depuis r est un arbre T couvrant G et vérifiant : pour tout sommet x de G on a $dist_T(r,x) = dist_G(r,x)$. On note $n_T(x)$ la valeur $dist_T(r,x)$.

Théorème (Arbre des plus courts chemins)

Un arbre T couvrant de G est un arbre des plus courts chemins depuis r si, et seulement si, pour toute arête xy de G, on a $|n_T(x) - n_T(y)| \leq 1$.

6- - Parcours en largeur (2/2)

- Un parcours en largeur est un parcours obtenu par application de l'algorithme suivant.

```
Algorithme : PARCOURS-EN-LARGEUR(G = (V, E), r)

pour tous les v ∈ V faire dv(v) ⟵ 0 ;                      // sommets déjà vus
dv(r) ⟵ 1 ; ordre(r) ⟵ 1 ; pere(r) ⟵ r ; niv(r) ⟵ 0 ;      // la racine
Enfiler r dans AT ;      // sommets à traiter, AT gérée comme une file
t ⟵ 2 ;                                                    // le temps
tant que AT ≠ ∅ faire
    Prendre v le premier sommet de AT l'enlever de AT ;
    pour tous les x ∈ Vois(v) faire
        si dv(x) = 0 alors
            dv(x) ⟵ 1 ;              // on traite x pour la première fois
            Enfiler x dans AT, en dernière position ;
            ordre(x) ⟵ t ; t ⟵ t + 1 ;
            pere(x) ⟵ v ;  niv(x) ⟵ niv(v) + 1 ;

retourner ordre, pere et niv
```

6- - Parcours en largeur (2/2)

- Un parcours en largeur est un parcours obtenu par application de l'algorithme suivant.

Théorème (Parcours en largeur)

L'appel PARCOURS-EN-LARGEUR$(G = (V, E), r)$ *retourne un arbre des plus courts chemins de* G *de racine* r*. Sa complexité est en* $O(n+m)$*.*

7- Parcours en profondeur (1/2)

- Soit $T = (V, E)$ un arbre et r un sommet de T choisi comme racine. Les ancêtre d'un sommet x de T sont tous les sommets de l'unique chemin de T reliant r à x.
  La branche issue de x dans T est l'ensemble des sommets de T qui admettent x comme ancêtre.

7- Parcours en profondeur (1/2)

- Soit $T = (V, E)$ un arbre et r un sommet de T choisi comme racine. Les ancêtre d'un sommet x de T sont tous les sommets de l'unique chemin de T reliant r à x.
  La branche issue de x dans T est l'ensemble des sommets de T qui admettent x comme ancêtre.

- Un arbre couvrant T d'un graphe G est dit normal si pour toute arête xy de G, on a : x est dans la branche de T issue de y ou y est dans la branche de T issue de x.

7 - Parcours en profondeur (2/2)

- Un parcours en profondeur est un parcours obtenu par application de l'algorithme suivant.

Algorithme : PARCOURS-EN-PROFONDEUR($G=(V,E),r$)

pour tous les $x \in V$ faire $dv(v) \longleftarrow 0$; // sommets déjà vus
$dv(r) \longleftarrow 1$; $debut(r) \longleftarrow 1$; $pere(r) \longleftarrow r$; // la racine
Empiler r sur AT ; // sommets à traiter, AT gérée comme une pile
$t \longleftarrow 2$; // le temps
tant que $AT \neq \emptyset$ faire
    Noter x le sommet en haut de AT ;
    si $vois(x) = \emptyset$ alors
        Dépiler AT ;
        $fin(x) \longleftarrow t$; $t \longleftarrow t+1$; // fin de traitement pour x
    sinon
        Noter y le sommet en haut de $vois(x)$ et dépiler $vois(x)$;
        si $dv(y) = 0$ alors
            $dv(y) \longleftarrow 1$; // on traite y pour la première fois
            Empiler y sur AT ;
            $debut(y) \longleftarrow t$; $t \longleftarrow t+1$;
            $pere(y) \longleftarrow x$;

7 - Parcours en profondeur (2/2)

- Un parcours en profondeur est un parcours obtenu par application de l'algorithme suivant.

Théorème (Parcours en profondeur)

L'appel PARCOURS-EN-PROFONDEUR$(G=(V,E),r)$ *retourne un arbre normal de* G *de racine* r*. Sa complexité est en* $O(n+m)$*.*

8- Algorithme de Dijkstra (1/2)

- On considère un graphe $G = (V, E)$ donné par liste de voisin avec l une fonction de longueur positive sur les arêtes, et r un sommet de G, la racine.
- On veut calculer une fonction $d : V \to \mathbb{R}^+$ donnant la distance à la racine r et une fonction $pere : V \to V$ codant l'arbre des plus courts chemins correspondant.

8- Algorithme de Dijkstra (1/2)

- On considère un graphe $G = (V, E)$ donné par liste de voisin avec l une fonction de longueur positive sur les arêtes, et r un sommet de G, la racine.
- On veut calculer une fonction $d : V \to \mathbb{R}^+$ donnant la distance à la racine r et une fonction $pere : V \to V$ codant l'arbre des plus courts chemins correspondant.
- Un algorithme 'en bricolage' : on remplace chaque arête par une ficelle de la longueur correspondante et on cloue la racine au mur. Les ficelles droites donnent l'arbre recherché !

8- Algorithme de Dijkstra (2/2)

Algorithme : $\textsc{Dijkstra}(G = (V, E), r)$

pour tous les $v \in V$ faire
- $d(v) \longleftarrow +\infty$;
- $traite(v) \longleftarrow 0$; // pour marquer les sommets traités

$pere(r) \longleftarrow r$; $d(r) \longleftarrow 0$; // la racine

tant que *il existe* x *avec* $traite(x) = 0$ faire
- Choisir un tel x avec $d(x)$ minimum ;
- $traite(x) \longleftarrow 1$;
- pour tous les $y \in Vois(x)$ faire
  - si $traite(y) = 0$ *et* $d(y) > d(x) + l(xy)$ alors
    - $d(y) \longleftarrow d(x) + l(xy)$; // x est un raccourci pour atteindre y
    - $pere(y) \longleftarrow x$;

8- Algorithme de Dijkstra (2/2)

Théorème (Algorithme de Dijkstra)

L'appel $\textsc{Dijkstra}(G = (V, E, l), r)$ *retourne un arbre des plus courts chemins valués de* G *de racine* r*. Sa complexité est en* $O(n^2)$*.*

- Si on gère 'la frontière' des sommets traités (c-à-d les sommets v avec $traite(v) = 0$ et $dist(v) < +\infty$) par un tas alors on obtient une complexité en $O(m \log n)$.