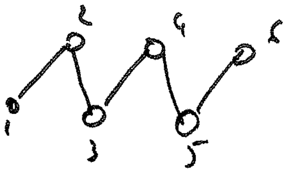
Chap I : Déf de bases.

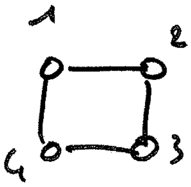
I] Graphes.

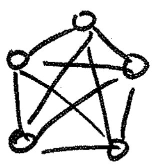
- Un graphe $G=(V,E)$ est constitué :
  - d'un ensemble de sommets V, de taille n
  - ——— arêtes $E\subseteq V^2$, ——— m.

Deux sommets $x,y\in V$ tels que $xy\in E$ sont voisins, reliés ou encore adjacents. Les arêtes sont non orientées (non), on écrit $xy\in E$ ou $yx\in E$.

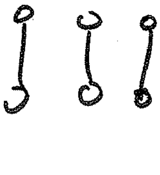
- Exemples : (sous ens. de taille 2 de V avec joli dessin).

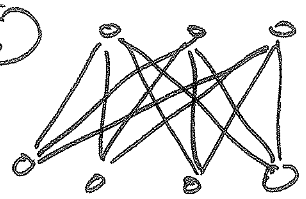
① 
$V=\{1,2,3,4,5,6\}$
$E=\{12,23,34,45,56\}$
Chemin de longueur 5 noté $\boxed{P_5}$.

② 
$V=\{1,2,3,4\}$
$E=\{12,23,34,41\}$
Cycle de taille 4 noté $\boxed{C_4}$

③ 
Graphe complet (toutes les arêtes possibles) ou clique de taille 5 noté $\boxed{K_5}$.

④ 
Stable de taille 3 (pas d'arêtes).

⑤ 
Couplage : arêtes deux à deux disjointes

6 

Graphe biparti complet $K_{3,4}$

// Modèles pour : réseaux, graphe de conflit, interconnections....

Rem : . Un graphe est biparti si il admet une partition de ses sommets en deux stables (G_6 est biparti, pas G_5 → exo).

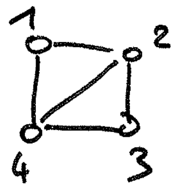
. Sauf contraire, on interdit les boucles (arête xx) 

et les arêtes multiple (pls arêts xy) 

II) Codage

- On code généralement V par $\{1, \dots, n\}$ ou $\{0, \dots, n-1\}$
- On peut coder généralement de 3 façons :
  1. Par liste d'arêts
  2. Par liste de voisins : on associe à tout $v \in V$ la liste $L(v)$ de ses voisins
  3. Par matrice d'adjacence :

$$A_G = (a_{ij})_{1 \leq i,j \leq n} \quad \text{où} \quad a_{ij} = \begin{cases} 0 & \text{si } ij \notin E \\ 1 & \text{si } ij \in E. \end{cases}$$

Ex : $G =$ 

① On code G par $\{12, 14, 24, 23, 34\}$

② ———— $L(1) = \{2,4\}$, $L(2) = \{1,3,4\}$
$L(3) = \{2,4\}$, $L(4) = \{1,2,3\}$

③ ———— $A_G = \begin{pmatrix} 0&1&0&1 \\ 1&0&1&1 \\ 0&1&0&1 \\ 1&1&1&0 \end{pmatrix}.$

- Propriétés de A_G :
  - A_G symétrique car G non orienté ($xy \in E \iff yx \in E$)
  - Diagonale de 0 car G sans boucle.

Avantages :	Liste d'arêtes	Liste voisins	Matrice d'adj.
Taille codage	$O(m)$	$O(n+m)$	$O(n^2)$
Temps pour décider "$xy \in E$" ?	$O(m)$ // ou $O(\log n)$ si recherche dicho	$O(n)$ // ou $O(\log n)$ si triées	$O(1)$ (ie : temps cst) (on suppose l'accès mémoire cst)
Trouver les voisins de x	$O(m)$	$O(\lvert L(x)\rvert)$	$O(n)$.

Rq : . On a : $m \leq \frac{n(n-1)}{2} \rightsquigarrow m = O(n^2)$.

. On néglige la taille du codage des entiers : on suppose que $\{1, .., n\}$ prend place $O(n)$ et pas $O(n \log n)$.

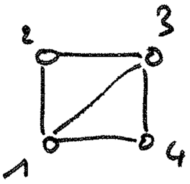
. C'est le temps du "pire des cas"

// lequel choisir : ça dépend ce qu'on veut faire, si on a de la place mémoire prendre liste voisin + mat d'adj.

III) Degrés.

. Le **degré** d'un sommet $v \in V$ est le nombre de ses voisins, on le note **$d(v)$**. C'est le nombre d'arêtes **incidentes** à v (ie : dont une extrémité est v)

Exemple :



$d(1) = 3$ $\quad d(3) = 3$ $\quad d(1)+d(2)+d(3)+d(4) = 10$.

$d(2) = 2$ $\quad d(4) = 2$

. **Formule des degrés** Pour tout $G = (V, E)$ $\sum_{v \in V} d(v) = 2 \cdot |E|$.

Pr : chaque arête xy est compté 1 fois dans $d(x)$ et 1 fois dans $d(y)$. ■

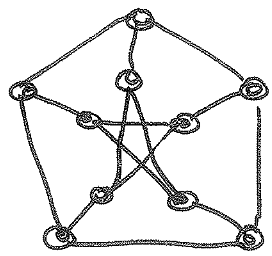
. **Remarques** : . la somme des degrés est toujours paire.

. Un graphe est **k-régulier** si tous ses sommets ont degré exactement k : 0-régulier → stable

1-régulier → couplage

- 2-régulier → union disjointe de cycle 
- 3-régulier $\overset{=\text{cubique}}{}$ → ça devient + compliqué structurellt.

Un célèbre :

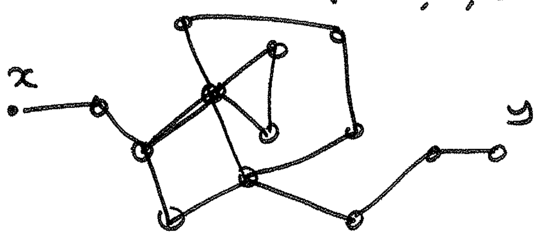


le graphe de <u>Petersen</u> . P
(plus petit cubic classe 2 = snark, plus petit cubic non hamiltonien, plus grand $\Delta = 3$ $\phi = 2$: Moore)

IV Marches et composantes connexes.

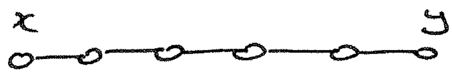
- Soit $x, y \in V$. Une <u>xy-marche</u> de G est une suite $x = x_0, x_1 \dots x_\ell = y$ telle que $x_i x_{i+1} \in E$ $\forall i = 0, \dots, \ell - 1$.

<u>Ex</u> :



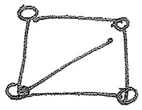
Sa <u>longueur</u> et ℓ

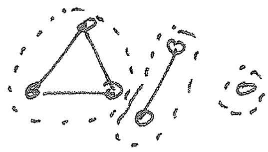
- Si tous les x_i sont disjoints alors on a un <u>xy-chemin</u>

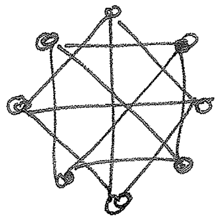


- On va étudier : | <u>ENTRÉE</u> : G graphe, x, y sommets de G.
  | <u>SORTIE</u> : VRAI si il existe une xy-marche dans G, FAUX sinon.
- Rq : Pbl de base en algo, connaît de multiples extensions.
- On introduit la relation $x \sim y$: "il existe une xy-marche dans G"
  Cette relation vérifie (<u>exo</u>) :
  (i) $x \sim x$ (réflexive)
  (ii) $x \sim y \Leftrightarrow y \sim x$ (symétrique)
  (iii) $x \sim y$ et $y \sim z \Rightarrow x \sim z$ (transitive)

C'est une relation d'équivalence, ses classes sont les <u>composantes connexes</u> de G. Autrement dit, ce sont les + gdes parties de G "d'un seul tenant"

- Exemple : . $G =$  1 seule composante

  . $G =$  3 composantes.

  . G :  2 composantes (mais moins visibles...)

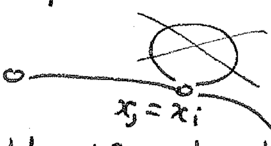
- Si G a une seule composante connexe, il est dit <u>connexe</u>.

- <u>Lemme</u> : G contient une xy-marche $\Leftrightarrow$ G contient un xy-chemin.

<u>Pr</u> : $\Leftarrow$: immédiat

$\Rightarrow$: on considère M une xy-marche sur laquelle on applique l'algo. suivant : <u>tant que</u> M contient 2 sommets identiques : $x_i = x_j$

$\quad M \leftarrow x_1 x_2 \ldots x_{i-1} x_i x_j x_{j+1} \ldots x_\ell$ (on raccourcit M)



La taille de M décroit strictement à chaque boucle donc l'algo termine. La structure résultante est un chemin. ■

- <u>Rq</u> : les comp. connexes sont aussi les classes d'éq. de $x \sim y$: "il existe un xy-chemin dans G".

<u>V</u>] <u>Calcul des composantes</u>.

On veut résoudre :

- <u>ENTRÉE</u> : $G = (V, E)$ donné par une liste d'arêtes.
- <u>SORTIE</u> : une fonction $comp : V \to V$ telle que $comp(x) = comp(y)$ ssi il existe une xy-marche (ou un x-y chemin ...)

// Une fois ça calculé, on peut répondre à "$\exists$ xy-marche ?" en temps cst.