dépile père(y), on place y en fin de file, puis lorsqu'on dépile père(x), x s'empile derrière y. Donc y sera traité avant x et ordre(y) < ordre(x), ce qui est exclu □

- <u>Fait 2</u> : ordre(x) < ordre(y) ⇒ niv(x) ≤ niv(y).

<u>Preuve</u> : par récurrence sur le nombre de sommets traités :

H_k : "$\forall x, y$ ordre(x) ≤ k et ordre(y) ≤ k alors $o(x) < o(y) \Rightarrow n(x) \leq n($

C'est vrai par k = 2 : on a la racine r et un sommet x de niveau 1

Si H_k est vraie : prenons z : ordre(z) = k+1

Soit x avec ordre(x) < ordre(z), on a par le fait 1

ordre(père(x)) ≤ ordre(père(z)) et tous deux sont ≤ k

donc $n_T(\text{père}(x)) \leq n_T(\text{père}(z))$ soit $n_T(x) - 1 \leq n_T(z) - 1$

et $n_T(x) \leq n_T(z)$, on a H_{k+1}. □

- <u>Fait 3</u> : Pour toute arête xy de G, on a : $|n_T(x) - n_T(y)| \leq 1$.

<u>Preuve</u> : On pose ordre(x) < ordre(y) (sinon, on échange x et y).

Lorsqu'on dépile x :

- Si dv(y) = 0 alors père(y) = x et $n_T(y) = n_T(x) + 1$ OK.
- Si dv(y) = 1, c'est que le père de y a déjà été calculé et dépilé : (y étant dans la pile)

ordre(père(y)) < ordre(x) < ordre(y)

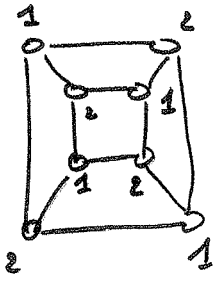
donc $n_T(\text{père}(y)) \leq n_T(x) \leq n_T(y)$

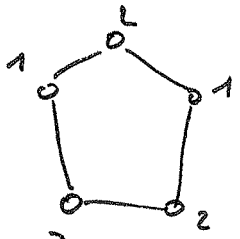
$n_T(y) - 1 \leq n_T(x) \leq n_T(y)$. ■

Problème du jour n°4

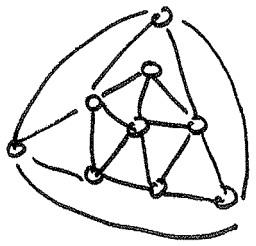
- Un graphe $G = (V, E)$ est <u>k-colorable</u> si il existe une coloration des sommets de G avec les couleurs $1, \dots, k$ telle que pour toute arête xy, les couleurs de x et y sont différentes.

- <u>Exemple :</u>





3 cycles impairs.



4 couleurs (planaire !)

- <u>Rem :</u> tous les sommets d'une même couleur forment un stable : on cherche donc à partitionner le graphe en un nombre min de stable.
  - Bipartis ⟺ partition en 2 stables ⟺ 2-colorable.
  - stable ⟺ 1-colorable.

NOMBRE CHROMATIQUE

<u>Entrée</u> : G un graphe
<u>Sortie</u> : trouver le minimum tel que G soit k-colorable.

NP-difficile... En fait, même le problème de décision suivant est dur :

3-COLORABLE

<u>Entrée</u> : G un graphe
<u>Sortie</u> : G est-il 3-colorable ?

NP-complet.

- On va regarder le problème de 2-coloration :

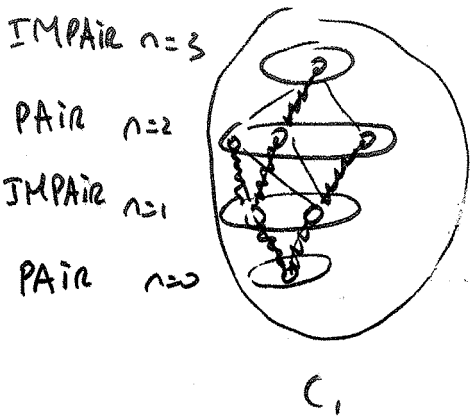
2-COLORABLE

| Entrée : G un graphe
| Sortie : G est-il biparti ?

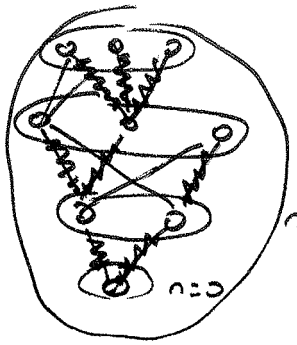
- G va être biparti si dans un parcours en largeur, on ne trouve jamais d'arête à l'intérieur d'un même niveau.

BIPARTI (G).

```
L1  [. Pour chaque composante connexe C de G.
L2      [. Calculer un parcours en largeur de C
L3      . Pour chaque arête xy dans C
L4          [. Si  n(x) = n(y) retourner FAUX.
L5  . Retourner VRAI.
```



C_1

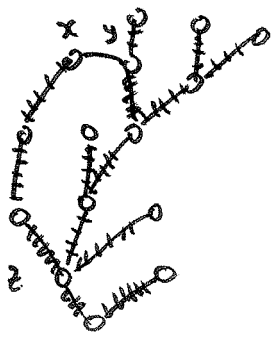


C_2

Si les arêtes ne sont que entre des niveaux consécutifs, la coloration PAIR-IMPAIR est une 2-coloration

Correction : . si l'algo renvoie vrai, voir ci-dessus.

. si l'algo renvoie faux. Il trouve une arête xy avec $n_T(x) = n_T(y)$:



Si on considère z le plus proche ancêtre commun de x et y dans T,

Les chemins xwz et ywz ayant même longueur, on trouve un cycle impair dans G, qui n'est donc pas 2-colorable ⊠

Rem: on a montré: G biparti $\Leftrightarrow$ G ne contient pas de cycle impair.

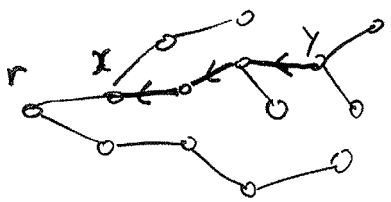
Complexité: $O(n+m)$ si on s'y prend bien...

V] Arbres normaux.

- But: trouver un arbre "étiré" depuis une racine choisie.
  courant

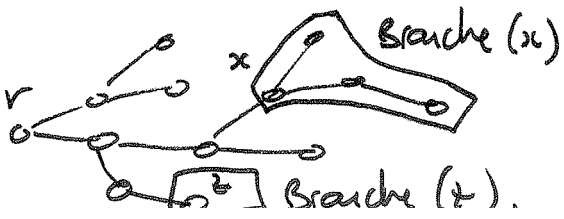
- Rem: trouver le plus étiré possible est illusoire, ça reviendrait à chercher un chemin hamiltonien depuis la racine.

- Etant donnés un arbre enraciné et une fonction père, x est un ancêtre de y si il existe une suite $x_1 = y, x_2, \dots, x_\ell = x$ telle que $\forall i = 2, \dots, \ell$ $\text{père}(x_i) = x_{i-1}$

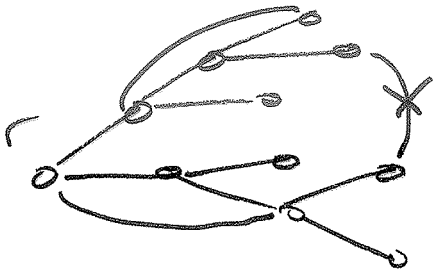


Autrement dit, x est sur le chemin de y à la racine.

- la branche issue de x est l'ensemble des sommets qui admettent x comme ancêtre.



• Un arbre couvrant enraciné T est <u>normal</u> si toute arête de G relie 2 sommets qui appartiennent à une même branche.

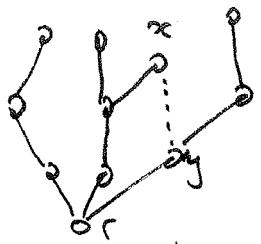


Le graphe "s'étire" le long des branches.

• <u>Lemme</u> : tout graphe connexe admet un arbre couvrant normal de racine arbitraire.

<u>Pr</u> : on choisit un arbre T avec $\sum_x n_T(x)$ le plus grand possible.

Si une arête xy relie 2 branches dans T :

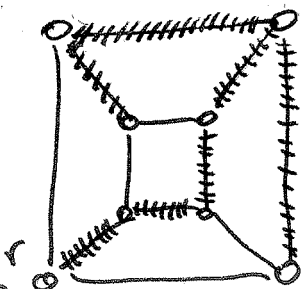


On peut supposer $n_T(x) \geq n_T(y)$

et dans $(T + xy) - y\,père(y)$ tous les sommets ont même niveau sauf les descendants de y qui ont gagnés au moins un niveau, ce qui contredit le choix de T. ∎

<u>Rq</u> : Ça donne un (mauvais) algo pour construire un arbre couvrant.

• <u>Exemple</u> :



On a intérêt à aller le plus loin possible pour que "les derniers sommets de l'arbre aient déjà leurs voisins comme ancêtre".

VI | Calcul effectif d'un arbre normal : parcours en profondeur

- On va utiliser une pile :

(comme les assiettes).

Premier Entré (LIFO).
Dernier Sortie

- On reprend l'algo de PARCOURS et on gère AT comme une pile. Comme les voisins d'un sommet ne sont pas tous traités consécutivement, on va aussi gérer les listes de voisin comme des piles. Enfin, on stocke deux dates (≠ ordres) : une date d'entrée dans AT : début et une date de sortie de AT : fin.

- PARCOURS PROFONDEUR (G, avec liste de voisins, r, la racine)

Initialisation.

```
L1   Pour tout v ∈ V dv(v) ← 0;
L2   dv(r) ← 1;
L3   Empiler r sur AT
L4   début(r) ← 1;  t ← 2;  //les dates.
L5   père(r) ← r
L6   Tant que AT ≠ ∅.
L7     x ← haut(AT)
L8     Si Vois(x) = ∅ alors
L9       Dépiler(AT);
L10      Fin(x) ← t;
L11      t ← t+1;
L12    Sinon
L13      y ← Haut(Vois(x));
L14      Dépiler(Vois(x));
```