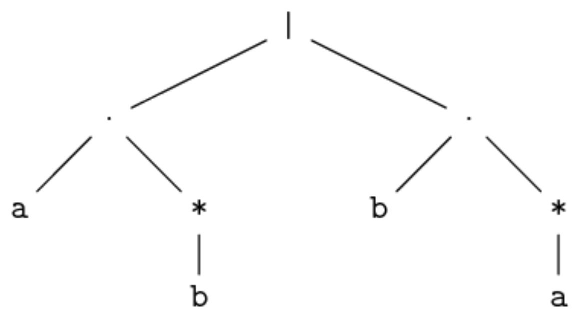
Théorie des Graphes
Arbres et Forêts

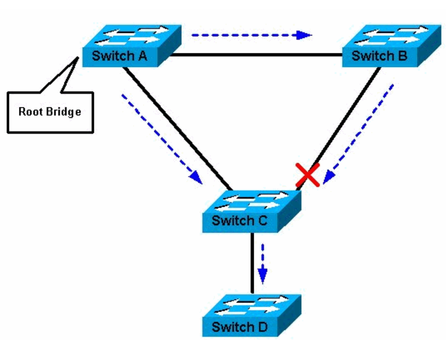
Fabrice Theoleyre

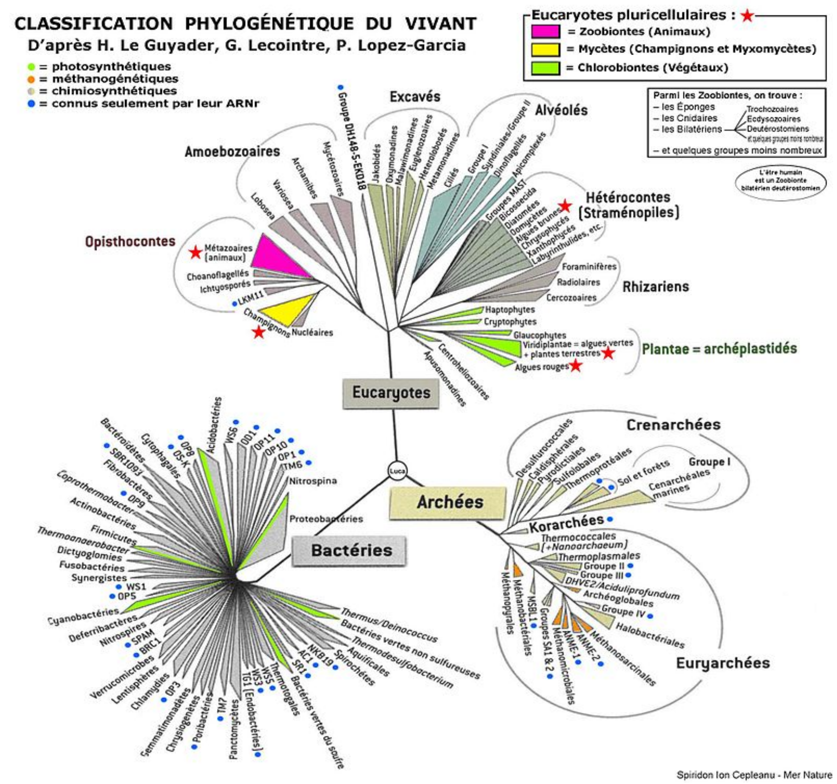
theoleyre@unistra.fr
http://www.theoleyre.eu

De l'utilité des arbres

- Arbres de décision
- Expressions régulières
- Les pannes dans Ethernet
- Phylogénie







Arbres & Forêts

- Soit G=(S,A) un graphe (non orienté simple). On dit que :
  - G est une forêt si G est sans cycle
    - Cycle = Une suite d'arêtes telle que je reviens à mon point de départ sur au moins un sommet
  - G est un arbre si c'est une forêt connexe
  - Dans les 2 cas, il s'agit d'une relation irréflexive (graphe sans boucle)

- Soit G=(S,A) un graphe tel que |A| >= 1
  - (i) si G est connexe alors |A| >= |S|-1
  - (ii) si G est sans cycle alors |A| <= |S|-1
  - (iii) si G est un arbre alors |A| = |S|-1
    - preuves en exercices !

Propriétés (en terme de connexité)

- Soit G=(S,A) un graphe avec au moins un sommet. Les assertions suivantes sont équivalentes :
  - (i) G est connexe et sans cycle (i.e. G est un arbre)
  - (ii) G est connexe et |A|=|S|-1
  - (iii) G est sans cycle et |A|=|S|-1
  - (iv) G est sans boucle et pour toute paire de sommets distincts de S, il existe une unique chaine entre eux.

- Preuves en exercices
  - en particulier (iii) => G est connexe

Arbres enracinés

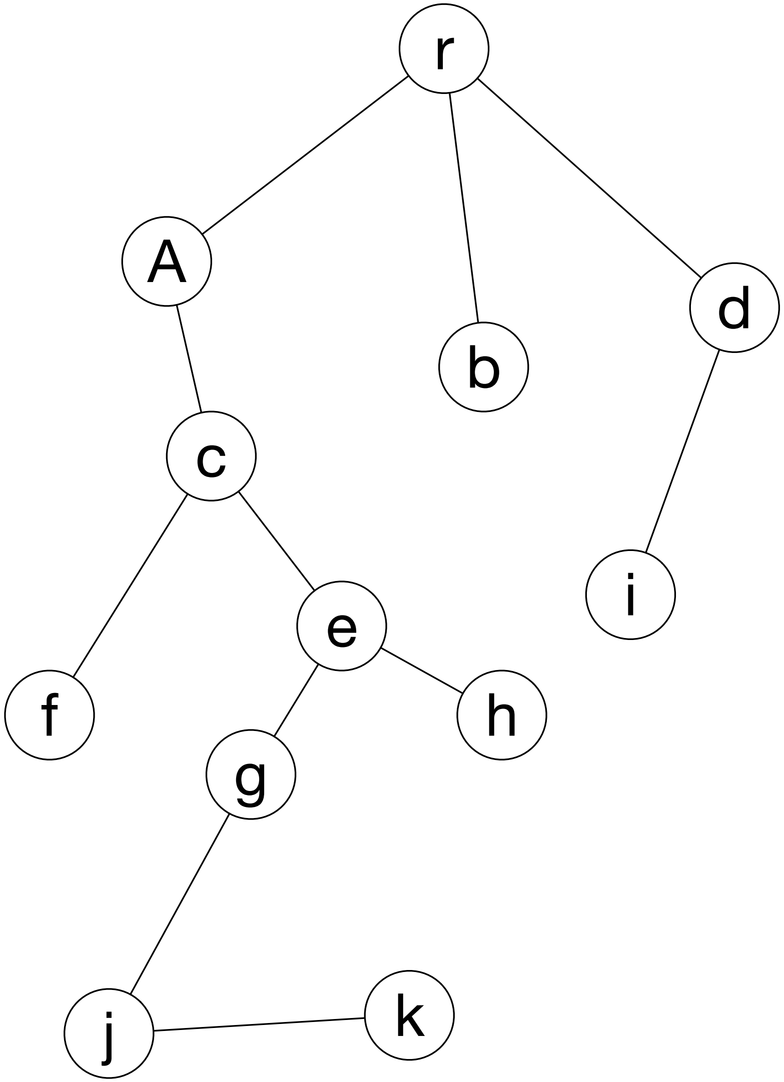
- Soit $G=(S,A)$ un arbre et $r \in S$
  - Le triplet $T=(S,A,r)$ représente l'arbre enraciné en r
  - On dit que r est la racine (root) de T
  - la terminologie des graphes reste identique (sommets, arêtes, etc)

- T un arbre enraciné en r
  - $x \in S$
  - ➔ il existe une unique chaine (élémentaire) entre r et x
  - Le chemin entre deux sommets quelconques du graphe est *unique*

Terminologie des arbres enracinés

- Père / fils (parent / child)
  - Soit 2 sommets x et y voisins dans T=(V,E,r).
    - ❖ si x appartient à la chaine de r à y, alors x est le père de y.
    - ❖ sinon x est le fils de y (y appartient à la chaine de r à x).
  - on peut généraliser à ancêtres/ascendant vs. Descendants

- Nœud / feuille (Node / Leaf)
  - x est un noeud (interne) ssi d(x)>=2 (ou x=r et d(r)=1)
    - ❖ interne → au moins un fils
  - x est une feuille ssi d(x)=1 (ou x=r et d(r)=0)
    - ❖ feuille → pas de fils



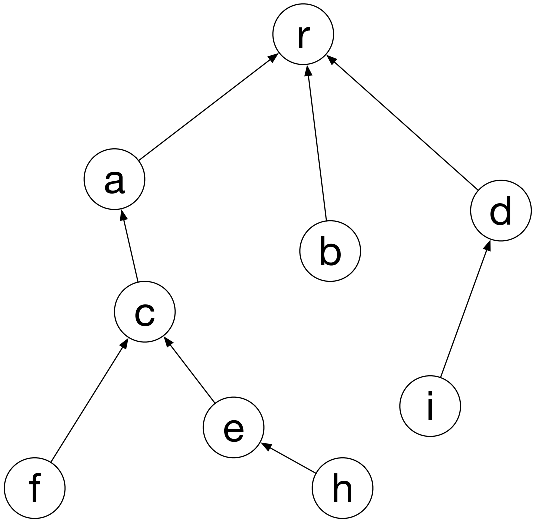
Terminologie des arbres enracinés (2)

- Sous-arbre
  - Soit T=(V,E,r) un arbre enraciné
  - Un sous arbre (enraciné) de T est un arbre enraciné (V',E',r') tel que
    - ❖ r' est un fils de r et (V',E') est une composante connexe du sous graphe engendré par V\{r}.
    - ❖ r' étant le seul voisin de r (dans T) appartenant à cette composante connexe
  - Par abus de langage, on appelle sous arbre tout arbre (enraciné) obtenu en répétant ce procédé.
  - Exemple en dessin !

- Hauteur
  - soit x un noeud appartenant à un arbre enraciné T=(V,E,r)
    - ❖ La hauteur de x dans T est la longueur de la plus petite chaine reliant r à x
    - ❖ la hauteur d'un arbre est la valeur maximale des hauteur des sommets.
  - Peut être appelée quelquefois « *profondeur* »

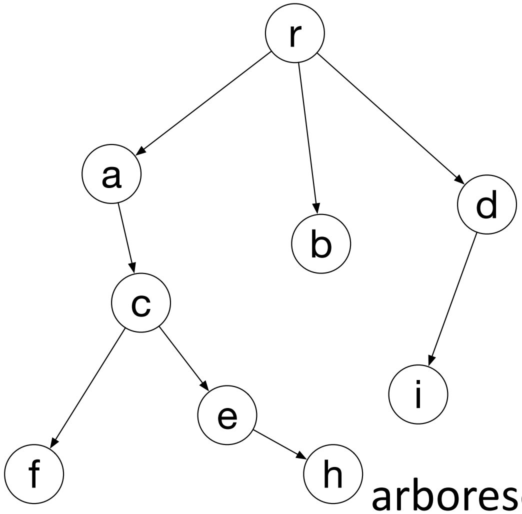
Arborescence

- **Arborescence**
  - Un graphe connexe, sans cycle, dont tous les sommets V sont descendants de r
    - ❖ Anti-arborescence : ascendants
  - Soit T=(V,E,r) un arbre enraciné -- (V,E) est non orienté
    - ❖ Une arborescence : T'=(V,E',r) telle que (V,E') est le graphe partiel obtenu en retirant de E les arêtes de la forme (x,y) où x est un fils de y dans T.
      - Le graphe (V,E) est la fermeture symétrique de (V,E')
      - Relation binaire R sur X → $S = R \cup \{(x,y) \mid (y,x) \in R\}$
        - » Union de R et de son inverse R^-1



anti-arborescence

- **Remarques :**
  - on utilise souvent le mot arbre pour désigner une arborescence
    - ❖ (ou juste un arbre enraciné)
  - on peut descendre mais pas remonter
  - les notions précédentes sont tjs valables : fils = successeur, etc.

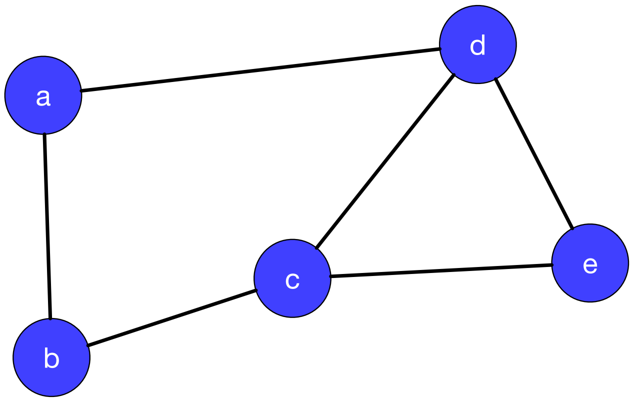


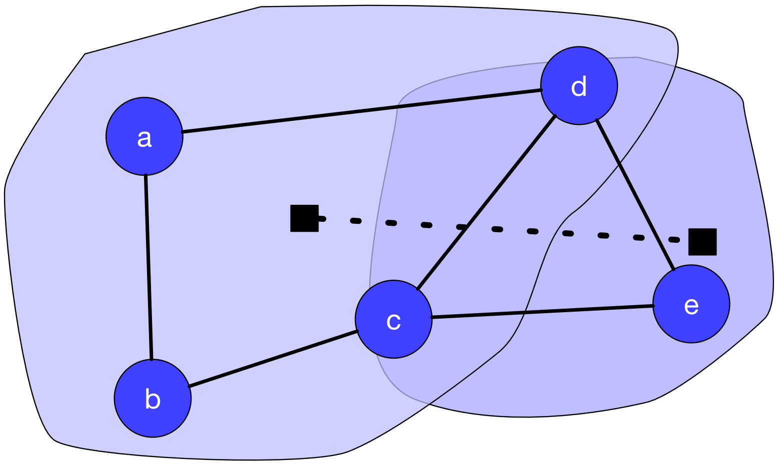
arborescence

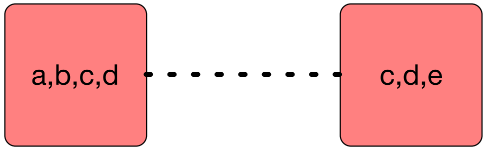
- **Propriété : Soit T=(V,E,r) une arborescence**
  - On a $d^-(r)=0$ et $d^-(x)=1$ pour tout $x \neq r$

Largeur arborescente (*Tree-Width*)

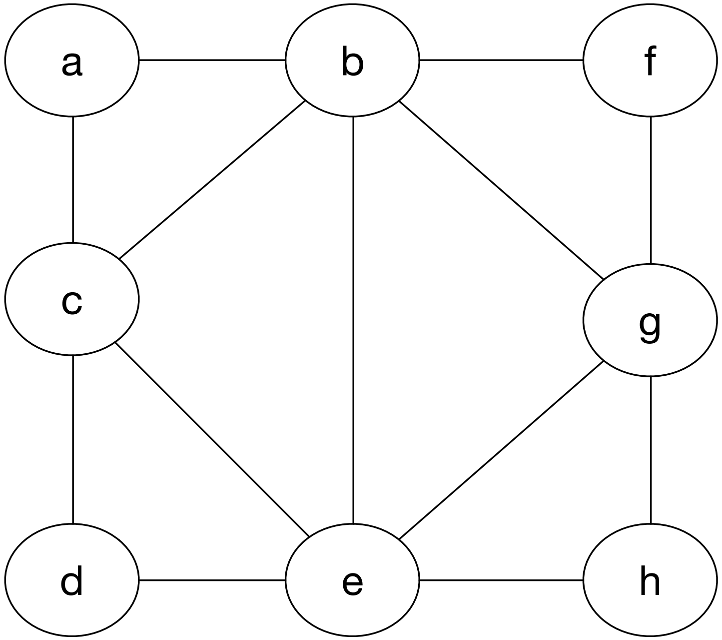
- Décomposition arborescente
  - Décomposition de G en sous-ensembles (= étiquette $\lambda(t)$)
    - Associe à chaque étiquette t un sous-ensemble de sommets $\lambda(t)$
    - $\cup_{t\in T} \lambda(t) = V$
    - Attention, $\cap_{t\in T} \lambda(t) = \emptyset$ est **faux** dans le cas général
  - T un arbre
  - Pour chaque arête (u,v), il existe une étiquette qui contient u et v
    - $\cup_{t\in T} \lambda(t)\, \forall (u,v) \in E, \exists t \in T |\, (u,v) \in \lambda(t)$
  - Pour tout sommet v, le sous-ensemble des étiquettes auxquelles il appartient forme une composante connexe
    - $\{t \in T\, | v \in \lambda(t)\}\, connexe$
- Largeur arborescente
  - Plus petite largeur de toutes les décompositions possibles
    - Cardinal de la plus grande étiquette – 1
    - $Max_{t\in T} |\lambda(t)| - 1$
  - Calcul NP-difficile
- Métrique qui mesure la *proximité* par rapport à un arbre
  - Beaucoup de problèmes « *faciles* » sur les arbres



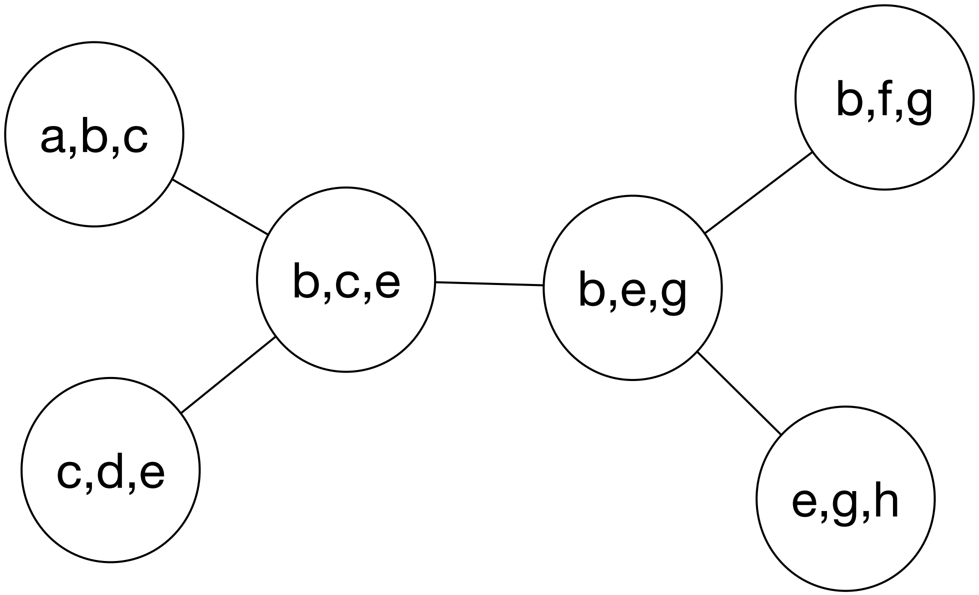




Décomposition arborescente



Largeur arborescente = 2

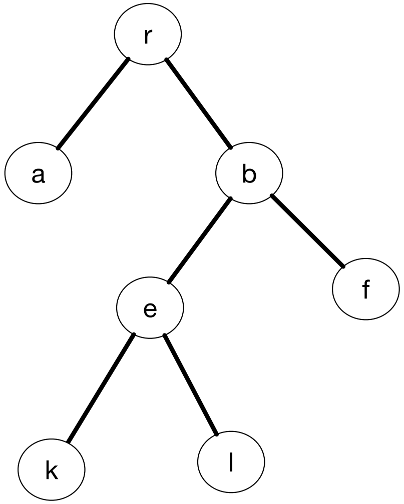


ARBRES K-AIRES

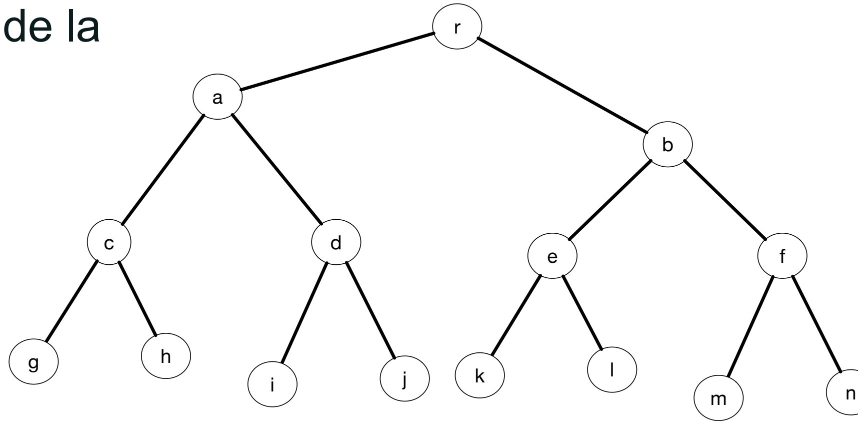
B-tree, etc.

Arbres k-aires

- Arborescence k-aire
  - Arbre enraciné dont chaque sommet a au plus k fils
  - k=2 → arbre binaire



- Définitions
  - Niveau
    - ❖ Ensemble des sommets équidistants de la racine
    - ❖ À même hauteur (iso-ligne)
  - Arbre k-aire localement complet
    - ❖ Un sommet possède soit k fils, soit 0
  - Arbre k-aire complet
    - ❖ Les feuilles sont toutes à la même hauteur



Utilité des arbres binaires

- Arbres binaires de recherche (b-tree)
  - Algorithmes naturellement récursifs
  - Utiles en bases de données
- Stockage mémoire
  - Liste chainée

```
typedef struct{
        elem* gauche;                    // si gauche & droit NULL → feuille
        elem* droit;
        void * val;
}elem;
elem *graphe;   //élement racine
```

- Création de liste triée
  - Si élément plus petit → insertion à gauche
  - Si élément plus grand → insertion à droite
  - Ajoute au premier pointeur NULL trouvé
- Complexité
  - Ajout en O(log n) en cas moyen, au pire O(n)
  - Recherche en O(log n) en cas moyen, au pire O(n)

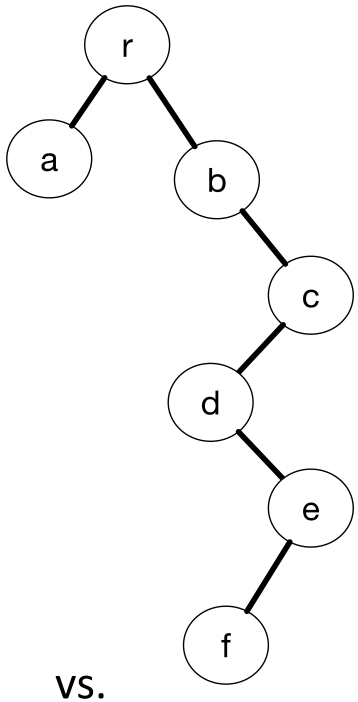
Pseudo-Code

```
void addVal(elem**tree, void *val) {
   //on ajoute l'élément, c'est une feuille
  if (*tree == NULL)
     elem *e
     e->val = val;
     e->left = NULL;
     e->right = NULL;
     *tree = e;
     return();
   }
   //on insère à gauche ou à droite
   if (pval < elem->val)
       addVal(&(elem->gauche), val);
   else
       addVal(&(*elem->droite), val);
}
```

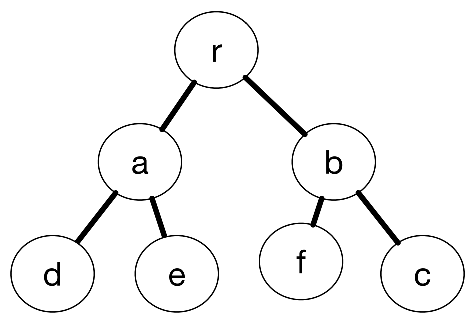
```
void rechSommet(elem *tree, void *val) {
   if (tree == NULL)
     return(NULL);
  if (cmp(val, tree->val) == 0)
     return(tree);
   if (cmp(val, tree->val) <0)
     return(rechSommet(tree->gauche,val));
  if (cmp(val, tree->val) >0)
     return(rechSommet(tree->droite,val));
}
```

De la « *bonne forme* » d'un arbre binaire

- Arbre équilibré
  - Différence de hauteur entre gauche et droite d'au + 1
- Arbre parfaitement équilibré
  - Différence de cardinalité des sous-arbres de gauche et droite d'au + 1
- Arbre dégénéré
  - Tous les nœuds ont 0 ou 1 fils
    - ❖ Arbre filiforme
- Maintenir une faible hauteur
  - Pourquoi ?
    - ❖ Parcours de l'arbre jusqu'à une feuille
  - Objectif : Maintenir la hauteur à $\lceil \log_2(n) \rceil$
    - ❖ à une constante près

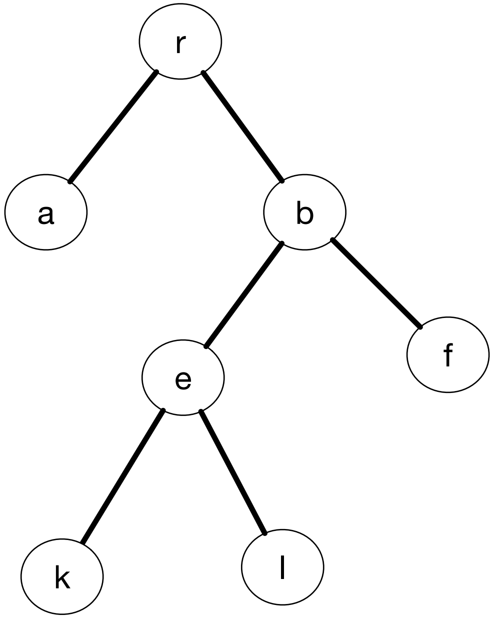


vs.



Parcours des arbres binaires

- 3 méthodes spécifiques aux arbres binaires
  1. Préfixe
     - ❖ Racine, fils gauche, fils droit
  2. Postfixe (ou suffixe)
     - ❖ Fils gauche, fils droit, Racine
  3. Infixe
     - ❖ Fils gauche, racine, fils droit
     - ❖ Ex : ((0, a, 0), r, (((0,k,0) , e , (0,l,0)), b, (0, f, 0))
     - ❖ Exercice : prouver que *le parcours infixe d'un b-tree donne une liste ordonnée de valeurs sous forme croissante*



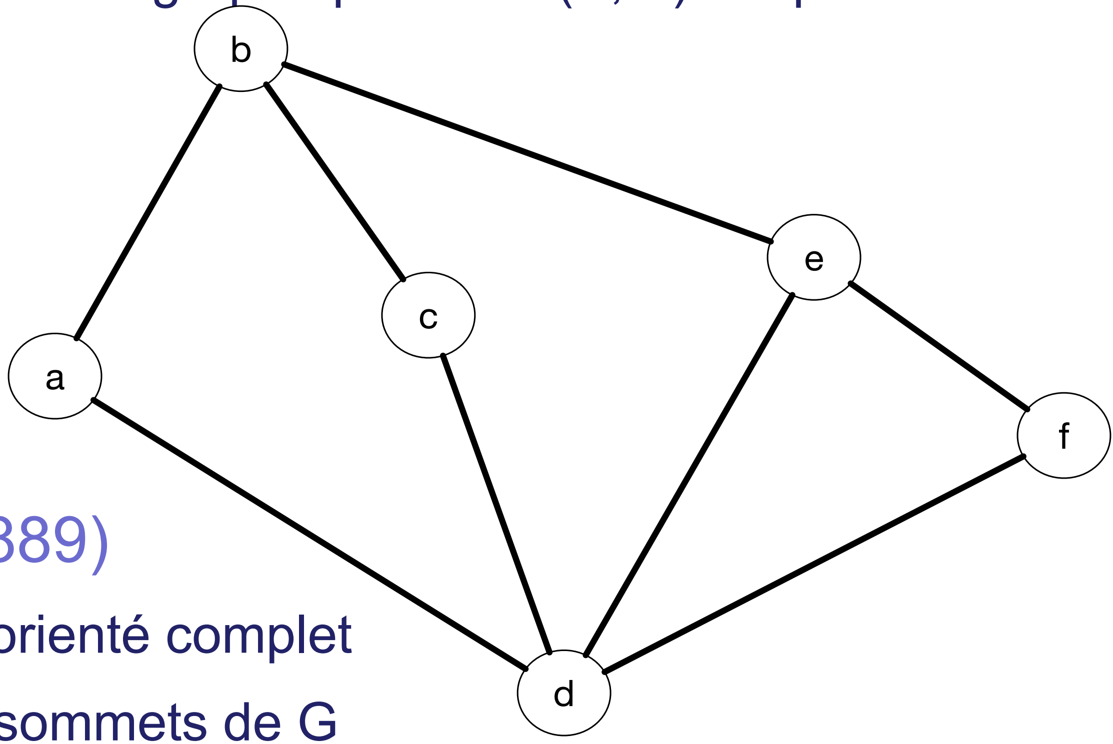
- … et toujours les parcours en largeur et en profondeur
  - Comment implémenter un parcours en largeur ?
    - ❖ Files d'attente (FIFO)

ARBRES COUVRANTS

De poids min / max

Arbres couvrants

- Arbre qui comporte tous les sommets
  - G=(V,E) un graphe non orienté connexe
  - Un arbre couvrant de G est un graphe partiel T=(V,E') tel que T soit un arbre



- Propriété (A. Cayley, 1889)
  - G=(V,E) un graphe non orienté complet
  - Soit n=|V| le nombre de sommets de G
  - Il existe n^{n-2} arbres couvrants de G.

Arbres couvrants de poids extremum

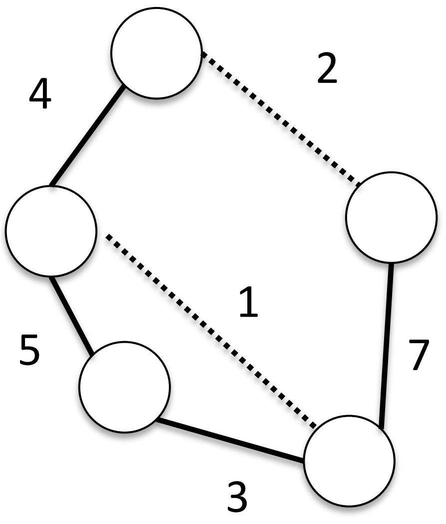
- Graphe pondéré
  - Poids (réel) associé à chaque arête
  - Poids d'un graphe G=(V,E) :
    - ❖ $P(G) = \sum_{(x,y)\in E} P(x,y)$
      - – $P(x,y)$ dénotant le poids de l'arête (x,y)

- Arbre couvrant de poids minimum
  - Minimum Spanning Tree (MST)
  - graphe partiel $T_{min}(V, E')$ de G=(V,E,P), tel que :
    - ❖ T_{min} est un arbre couvrant de (V,E)
    - ❖ pour toute arbre couvrant $T' \in ST$
      - – $\forall T' \in ST, P(T_{min}) \leq P(T')$
  - Arbre couvrant de poids maximum
    - ❖ $\forall\, T' \in ST, P(T_{max}) \geq P(T')$

Propriétés

- Un arbre couvrant de poids minimum pour le graphe G=(S,A,P) = un graphe de poids maximum pour G'=(S,A,-P)

- Soit G=(S,A,P) un graphe pondéré (non orienté, connexe et irréflexif) et soit T=(S,A') un arbre couvrant de G
  - Soit a = (x,y) ∈ A \ A' une arête de G
    - On note $C_a \subseteq A$ l'ensemble des arêtes de G constituant l'unique chaine de x à y dans T.
    - $\forall a \in A \setminus A', P(a) \leq \min_{c \in C_a} \mathrm{P(c)}$ ssi T est un arbre couvrant de poids **maximum**
  - Soit b ∈ A' une arête de T
    - On note $\Omega_b \subseteq A / A'$ l'ensemble des arêtes de A / A' ayant leurs extrémités dans les deux composantes connexes de (S, A'\{b}).
    - $\forall b \in A', P(b) \geq \max_{c \in \Omega_b} \mathrm{P(c)}$ ssi T est un arbre couvrant de poids maximum
  - Ces deux assertions sont donc équivalentes.
    - preuves en exercice



Algorithmes de calculs de MST

- Trouver une méthode *efficace* de calcul d'arbre de poids extremum
  - Ces calculs sont "équivalents" qu'il s'agisse du poids max ou min

- Recherche exhaustive
  - Calcule le poids de chaque arbre
  - Coût ?
    - Calcul du poids d'un arbre : somme de ses arêtes, en O(E)
    - Combien d'arbres ?
      - Pire cas du graphe complet de k sommets
      - k^(k-2) arbres
    - K= 50 → 50^48 = 3,5 *10^81
    - Irréaliste…

Kruskal v1 : version additive

- Soit G=(S,A,P) un graphe pondéré (non orienté, connexe et sans boucle)
  - n=|S| et m=|A|
  - Soit $A = \{a_i\}_{i=1..m}$ tel que les arêtes a_i soient ordonnées par poids décroissant
    - $\forall i \in [1.,m], P(a_i) \geq P(a_{i+1})$
  - Soit $(A_i)_{i=0..m}$ la suite définie par
    - $A_0=\emptyset$
    - $\forall i \in [0,m-1],\ A_{i+1}=\begin{cases} A_i \cup \{a_{i+1}\} & si\ (S, A_i \cup \{a_{i+1}\})\ est\ sans\ cycle \\ A_i & sinon \end{cases}$

- Alors (S,A_m) est un arbre couvrant de poids maximum

- L'algorithme en découle directement, il se décompose en deux étapes
  1. tri des arêtes par ordre décroissant
  2. la construction de l'ensemble d'arêtes A_m, par ajouts successifs des éléments de la liste triée en vérifiant que cela ne crée pas de cycles dans le graphe partiel
     - On s'arrête dès que $|A_i|$=n-1
     - Nombre max d'arêtes dans un arbre (nb sommets -1)

Kruskal v2 : version soustractive

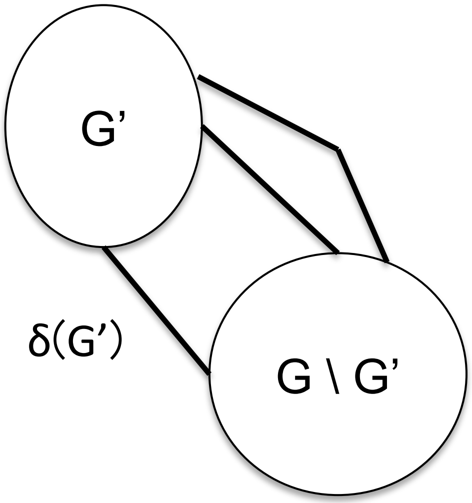
- On agit de façon symétrique
  - On supprimer cette fois les arêtes du plus petit poids au plus grand

- Formellement
  - G=(S,A,P) un graphe pondéré (non orienté, connexe et sans boucle)
  - Soit n=|S| et m=|A|
  - Soit $A = \{a_i\}_{i=1..m}$ tel que les arêtes a_i soient ordonnés par poids décroissant
    - ❖ $\forall i \in [1.,m], P(a_i) \geq P(a_{i+1})$
  - Soit $(A_i)_{i=0..m}$ la suite définie par
    - ❖ A_0=A
    - ❖ $\forall i \in [0, m-1],\ A_{i+1=} \begin{cases} A_i \setminus \{a_{i+1}\}\ si\ (S, A_i \setminus \{a_{i+1}\})\ est\ connexe \\ A_i\ \ sinon \end{cases}$

- L'algorithme version 2 est naturel
  - On supprime tant que le graphe reste connexe

Prim

- Arêtes frontières
  - Soit G=(S,A) un graphe non orienté
  - soit S'⊆S un sous ensemble de sommets de S engendrant le graphe G'=(S',A') sous-graphe de G
  - a=(x,y) est une arête frontière de G' ssi x ∈ S' et y ∉ S' (ou le contraire)
    - ❖ L'ensemble des arêtes frontières est noté δ(G')



- Soit G=(S,A,P) un graphe pondéré, non orienté, connexe et sans boucle
  - Soit n=|S| et m=|A|
  - Soit x un sommet de G
  - Soit la suite $G_i = (S_i, A_i)_{i\in[1..m]}$ par
    - ❖ $G_1 = (S_1, A_1) = (\{x\}, \emptyset)$
    - ❖ $G_{i+1} = (S_{i+1}, A_{i+1}) = (S_i \cup \{z\}, A_{i+1} \cup \{(y,z)\}$ tel que
      - – $P\big((y,z)\big) = \max\{\, P(a) |\, a \in \delta(G_i)\}$ et $y \in S_i, z \notin S_i$

- Alors $G_n=(S,A_n)$ est un arbre couvrant de poids maximum

Quel algorithme choisir ?

- Complexités temporelles des algorithmes
  - Naïvement : O(m^2)
  - Kruskal
    - ❖ tri : O(m log(m)) , quicksort
    - ❖ test pour l'union ou la suppression (version efficace) : O(n log(n))
    - ❖ **O(mlog(n))** au pire pour la version additive
    - ❖ $O(m \log(n)\ (\log\log(n^3)))$ en version soustractive
  - Prim
    - ❖ **O(m + n log(n))** avec les structures les plus efficaces

- Il existante des variantes plus compliquées pour le calcul d'une arborescence de poids min / max
  - Mais toujours polynomial

Variante de l'arbre de Steiner

Définition

- Soit un graphe G(S,A)
- Soit un sous-ensemble de sommets $S' \subseteq S$
- T(S'', A') un sous-graphe de G
  - ❖ $S'' = S' \cup \{u\}_{u \in S \mid (u,v) \in A'}$
  - ❖ T est connexe
  - ❖ $\sum_{e \in A'} p(e)$ est de poids minimum
- Problème NP-Complet