ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
Section d'Informatique et de Systèmes de Communication

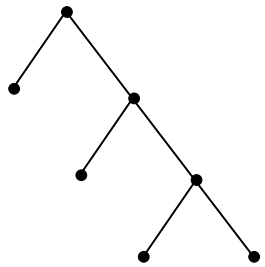
Corrigé de la série 4 12/10/2009

1. Graphes I

a) Si la première ligne de la matrice d'adjacence de G ne contient que des 1, alors le sommet x correspondant à cette ligne est connecté à tous les autres sommets du graphe. Le graphe est donc bien connexe puisqu'il y a un chemin entre toutes paires (y, z) de sommets de G (un chemin de longueur 1 si $x \in \{y, z\}$, sinon un chemin de longueur 2 qui passe par x).

b) Attention de ne pas confondre le *degré* d'un sommet (le nombre de fils) et le *facteur d'équilibre* (la différence entre les hauteurs de ses sous-arbres).

On peut prendre par exemple le graphe ci-dessous:



Tous les sommets ont degré 0 ou 2, pourtant la racine a comme facteur d'équilibre -2.

c) La preuve se fait par induction sur h. Écrivons N_h le nombre maximal de sommets que peut avoir un arbre. Un arbre binaire de hauteur h ne peut avoir qu'un seul sommet (la racine), donc $N_0 = 1 = 2^{0+1} - 1$.

Supposons maintenant l'affirmation prouvée pour h et montrons la pour $h+1$. Notons $\mathcal{T}_h$ l'ensemble d'arbres binaires de hauteur h. Alors comme tout arbre $T \in \mathcal{T}_{h+1}$ est formé d'une racine et de deux sous-arbres $L, R \in \mathcal{T}_h$ nous avons

$$\begin{aligned} N_{h+1} &= \max_{T \in \mathcal{T}_{h+1}} \{\#(\text{sommets de } T)\} \\ &= \max_{L,R \in \mathcal{T}_h} \{1 + \#(\text{sommets de } L) + \#(\text{sommets de } R)\} \\ &= 1 + \max_{L \in \mathcal{T}_h} \{\#(\text{sommets de } L)\} + \max_{R \in \mathcal{T}_h} \{\#(\text{sommets de } R)\} \\ &= 1 + (2^{h+1} - 1) + (2^{h+1} - 1) \\ &= 2^{h+2} - 1, \end{aligned}$$

ce qui termine la preuve.

d) • La plus grande hauteur que peut avoir T est $n - 1$. On peut le montrer très facilement par induction.

• La plus petite hauteur que peut avoir T est $\lfloor \log_2 n \rfloor$. Nous avons en effet montré dans la question c) que si T est un arbre de hauteur h avec n sommets alors

$$n \leq 2^{h+1} - 1.$$

En manipulant cette inégalité nous avons:

$$\begin{aligned} n \leq 2^{h+1} - 1 &\iff n + 1 \leq 2^{h+1} \\ &\iff \log_2(n+1) \leq h + 1 \\ &\iff h \geq \log_2(n+1) - 1, \end{aligned}$$

ainsi la plus petite hauteur que peut avoir h est

$$\lceil \log_2(n+1) - 1 \rceil .$$

Pour montrer que $\lceil \log_2 (n+1) - 1 \rceil = \lfloor \log_2 n \rfloor$, il suffit de montrer que

$$0 \leq \log_2 n - (\log_2 (n+1) - 1) \leq 1.$$

L'inégalité de droite est facile à vérifier. Pour l'inégalité de gauche, il suffit de prouver que

$$\log_2 (n+1) - \log_2 n \leq 1.$$

Ceci revient à prouver que

$$2^{\log_2 (n+1) - \log_2 n} \leq 2^1.$$

Or $2^{\log_2 (n+1) - \log_2 n} = \frac{n+1}{n} = 1 + \frac{1}{n}$, qui est bien inférieur à 2 pour $n > 1$.

e) Comptons plutôt le nombre de "demi-arêtes": Chacun des n sommets est connecté a d arêtes, il y a donc nd demi-arêtes (une demi arête n'est connectée qu'à un seul sommet). Puisqu'on a 2 demi-arêtes par arête, il doit y avoir un total de $\frac{nd}{2}$ arêtes.

Donc nd doit être divisible par 2 (le nombre d'arêtes est clairement un entier). Or si n et d sont impairs alors nd sera aussi impair. On en déduit que si n est impair alors est d doit être pair.

Une autre façon (très similaire) de procéder serait de regarder la matrice d'adjacence A de G. Dans chaque ligne, exactement d entrées ont comme valeur 1 (et $(n-d)$ entrées ont comme valeur 0). Ainsi comme il y a n lignes, le nombre total de 1's dans A est nd. Nous savons de plus les éléments sur la diagonale valent tous 0 (par hypothèse). Ces nd 1's se trouvent donc soit au dessus, soit en dessous de la diagonale.

Soit x le nombre de 1's au dessus de la diagonale et y le nombre de 1's en dessous de la diagonale (donc $x + y = nd$). Puisque G est non orienté, A est symétrique et donc $x = y$. Ainsi on a

$$nd = 2x,$$

et donc nd doit être pair. Or si n et d sont impairs alors nd sera aussi impair. On en déduit que si n est impair alors est d doit être pair.

2. Graphes II

a) **Proposition:** Un graphe G sans cycles avec n sommets et c composantes connexes a $n-c$ arêtes.

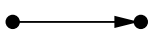
Preuve: Chaque composante de G est un graphe connexe (une seule composante) et sans cycles (puisque G est sans cycles). Donc chacune des c composantes forme un arbre. Soient $T_1, \ldots, T_c$ ces arbres, et soit n_i le nombre de sommets dans l'arbre T_i. Nous avons donc

$$\sum_{i=1}^{c} n_i = n.$$

En utilisant la formule d'Euler pour les arbres, nous voyons que T_i a $n_i - 1$ arêtes. Le nombre total d'arêtes dans G est donc

$$\sum_{i=1}^{c} (n_i - 1) = n - c.$$

b) i) On peut prendre par exemple:



ii) **Proposition:** Un graphe orienté $G = (V, E)$ fortement connexe avec $|V| \geq 2$ doit avoir au moins un cycle.

Preuve: Puisque $|V| \geq 2$, il existe deux sommets distincts $x, y \in V$ ($x \neq y$). Puisque G est fortement connexe, il existe:

- Un chemin $(x, c_1), (c_1, c_2), \ldots, (c_k, y)$ de x à y (longueur $k+1$).
- Un chemin $(y, d_1), (d_1, d_2), \ldots, (d_\ell, x)$ de y à x (longueur $\ell + 1$).

Donc le chemin $(x, c_1), \ldots, (c_k, y), (y, d_1), \ldots, (d_\ell, x)$ forme un cycle (de longueur $k + \ell + 2$).

3. Arbres

a) On obtient les suites suivantes pour les différents parcours:

Parcours	Suite
Preorder	I, D, A, C, B, G, E, F, H, O, M, K, J, L, N
Inorder	A, B, C, D, E, F, G, H, I, J, K, L, M, N, O
Postorder	B, C, A, F, E, H, G, D, J, L, K, N, M, O, I

b) Il s'agit de voir que pour chaque sommet r la propriété $g < r < d$ est vérifié pour tout sommet g dans le sous-arbre gauche et d dans le sous-arbre droit. Ceci est équivalent à voir que la suite obtenue par le parcours inorder est croissante (cf. point suivant). Ceci est manifestement vrai.

c) " $\Longleftarrow$ ". Nous montrons qu'un arbre de recherche résulte en une suite croissante si parcouru inorder.

Nous prouvons ce résultat par induction (forte) sur les arbres à n sommets. L'arbre vide ($n = 0$), correspond à la suite vide, qui est clairement croissante. Soit maintenant T

un arbre avec racine x, sous-arbre gauche L et sous-arbre droit R. La suite obtenue par parcours inorder est

$$[\text{suite du parcours de } L], x, [\text{suite du parcours de } R].$$

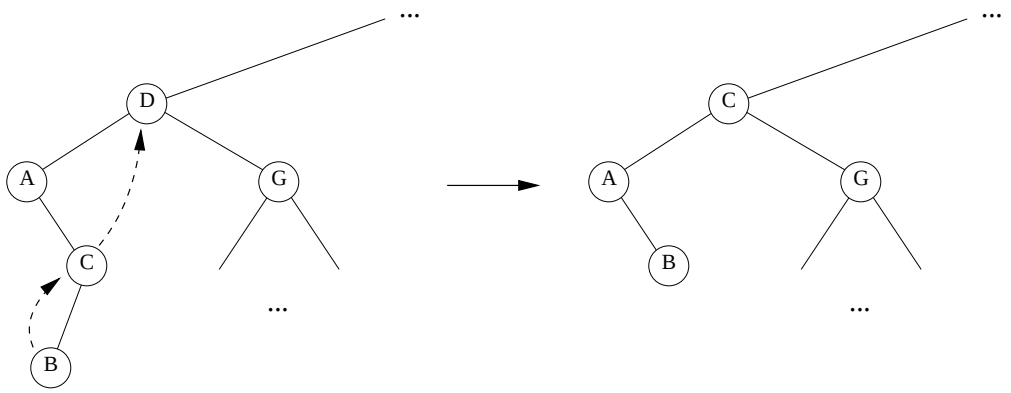
Par hypothèse d'induction, la partie L est croissante, et la partie R aussi. Il suffit donc de montrer que x est supérieur à son prédécesseur et inférieur à son succésseur. Comme T est un arbre de recherche, n'importe quel élément dans L est inférieur à x, donc aussi le dernier de la suite. De manière analogue, il n'y a pas non plus d'inversion entre x et son successeur.

" $\Longrightarrow$ ". Nous montrons que si le parcours inorder est croissant, alors l'arbre en question est un arbre de recherche. Soit x n'importe quel sommet dans l'arbre. Nous montrons que l'opération FIND(x) retourne bien le sommet x, ce qui permet de conclure.

Si x est racine de l'arbre de recherche, il sera clairement trouvé. Sinon, soit z n'importe quel sommet sur le chemin de la racine à x. Supposons sans perdre de généralité que le chemin descend dans le sous-arbre gauche de z, l'autre cas étant analogue. Alors x est parcouru avant z dans la traversée inorder, et donc $x < z$ par la croissance de la suite. Donc à l'étape où FIND se trouve en z, FIND descend aussi dans le sous-arbre gauche.

Comme cet argument s'applique à n'importe quel z sur le chemin vers x, FIND finit par trouver x.

d) On utilise l'algorithme du cours, i.e. on cherche dans le sous-arbre de gauche du sommet D le plus grand membre et on trouve C. Ce sommet ne peut pas avoir de sous-arbre droit (car ceci contredirait sa maximalité), ensuite, on remplace D par C et l'ancien sous-arbre gauche de C est mis à l'ancienne place de C. Schématiquement:



e) L'algorithme effectue plusieurs pas:

[1] Rechercher le sommet x à effacer
[2.0] S'il n'a pas de sous-arbre gauche, alors
    Effacer le sommet et mettre son sous-arbre droit à la place. stop.
[2.1] Trouver le plus grand élément du sous-arbre gauche de x:
$y \leftarrow$ `left`$[x]$
Tant que y a un sous-arbre droite
    $y \leftarrow$ `right`$[y]$

[2.2] Enlever y de l'arbre et mettre son sous-arbre gauche à sa place
[2.3] Enlever x de l'arbre et mettre y à sa place.

L'algorithme fonctionne parce que le sommet y vérifie la propriété que tous les autres sommets du sous-arbre gauche sont plus petits; ceci garantit que la propriété de l'arbre de recherche est présérvée. (Voir aussi page 72 des notes de cours).

4. Algorithmes récursifs: Le diamètre d'un arbre binaire

a) Notons r la racine de T. Pour un chemin dans T il y a trois possibilités:

(1) Le chemin se trouve entièrement dans T_1,

(2) Le chemin se trouve entièrement dans T_2,

(3) Le chemin passe par (ou se termine à) la racine r.

Pour pouvoir dire quelque chose sur le diamètre de T, nous considérons un chemin γ de longueur maximale. Si γ est du premier type, alors la longueur de γ est certainement égale à $\mathrm{diam}(T_1)$, et si γ est du deuxième type, sa longueur est égale à $\mathrm{diam}(T_2)$.

Le troisième cas nécessite une discussion un peu plus détaillée. Dénotons le dernier élément du chemin dans T_1 par x_1 et le dernier élément dans T_2 par x_2. Le chemin est alors comme suit:

$$x_1 \to \cdots \to \text{ racine de } T_1 \ \to \ \text{racine de } T \ \to \ \text{racine de } T_2 \ \to \cdots \to x_2.$$

La longueur de ce chemin est donc

$$\mathrm{profondeur}_{T_1}(x_1) + 1 + 1 + \mathrm{profondeur}_{T_2}(x_2),$$

où $\mathrm{profondeur}_T(x)$ dénote la profondeur du sommet x dans l'arbre T (voir p.57 du cours pour voir la définition de profondeur).

La maximalité du chemin implique maintenant que les deux termes $\mathrm{profondeur}_{T_1}(x_1)$ et $\mathrm{profondeur}_{T_2}(x_2)$ doivent être maximaux, i.e. que les sommets en question se trouvent en tout en bas de l'arbre et donc que la profondeur des sommets en question est égale à la hauteur du sous-arbre correspondant. En résumé donc, la longueur du chemin est, dans ce cas, égale à

$$h_1 + 2 + h_2,$$

où h_i dénote la hauteur de l'arbre T_i.

Nous déduisons donc que la longueur du plus long chemin dans T (et alors le diamètre de T) est égale à

$$\max\{\mathrm{diam}(T_1), \mathrm{diam}(T_2), h_1 + h_2 + 2\}.$$

Soit h la hauteur de T. Il est alors assez clair que

$$h = \max(h_1, h_2) + 1.$$

b) Nous construisons une fonction récursive qui calcule à la fois la hauteur et le diamètre d'un arbre binaire étant donné sa racine:

Call: HauteurEtDiam
Input: r: Racine d'un arbre binaire.
Output: (h, d): Hauteur et diamètre de l'arbre.
  if $r = 0$ **then**
    return $(-1, -1)$
  $(h_1, d_1) \leftarrow$ HauteurEtDiam(r[`left`])
  $(h_2, d_2) \leftarrow$ HauteurEtDiam(r[`right`])
  $h \leftarrow \max\{h_1, h_2\} + 1$
  $d \leftarrow \max\{d_1, d_2, h_1 + h_2 + 2\}$
  return (h, d)

Remarquons que cet algorithme a bien un temps de parcours linéaire: Si nous ignorons pour un instant les appels récursifs, nous voyons que, puisqu'il n'y a pas de boucles, l'algorithme opère en temps constant. Mais l'appel de la fonction HauteurEtDiam se fait exactement une fois pour chaque sommet; il en résulte que l'algorithme est $O(|V|)$.