Université des Sciences et de la Technologie Houari Boumediène
Faculté d'Electronique et d'Informatique
Département d'Informatique
LMD Master 1 RSD 2009/2010
Module ''Algorithmique avancée et complexité''

Travaux Dirigés
Série numéro 2 : Corrigé

Dénombrement sur les arbres binaires :

On note n le nombre de noeuds d'un arbre binaire, f son nombre de feuilles et h sa hauteur.

1. Quelle est la hauteur maximale d'un arbre à n noeuds ?

Avec n noeuds on construit un arbre de hauteur maximale quand chaque noeud, excepté la feuille, a un unique fils. L'arbre n'a alors qu'une seule branche qui contient les n noeuds et qui est de longueur n-1.

La hauteur maximale d'un arbre binaire est donc n-1 : $h \leq n-1$

2. Quel est le nombre maximal de feuilles d'un arbre de hauteur h ?

Si h = 0, c'est-à-dire si l'arbre se réduit à sa racine, f = 1.

Sinon, on raisonne par récurrence : le sous-arbre gauche (resp. droit) de la racine est un arbre de hauteur maximale h-1 et il a un nombre maximal de feuilles pour un arbre de hauteur h-1. Si on note f(h) le nombre maximal de feuilles d'un arbre de hauteur h on a donc :

f(h) = 1 si h = 0
2*f(h-1) sinon

On a donc f(h) =2^h.

Le nombre maximal de feuilles d'un arbre de hauteur h est donc 2^h : $f \leq 2^h$

3. Quel est le nombre maximal de noeuds d'un arbre de hauteur h ?

Si h = 0, c'est-à-dire si l'arbre se réduit à sa racine, n = 1. Sinon, on raisonne par récurrence : le sous-arbre gauche (resp. droit) de la racine est un arbre de hauteur h-1 et il a un nombre maximal de noeuds pour un arbre de hauteur h-1. Si on note n(h) le nombre maximal de noeuds d'un arbre de hauteur h on a donc :

n(h) = 1 si h = 0
1 + 2*n(h-1) sinon

On a donc n(h)=2^{h+1}-1

Le nombre maximal de noeuds d'un arbre de hauteur h est donc 2^{h+1}-1 : $n \leq 2^{h+1}-1$

4. Quelle est la hauteur minimale d'un arbre à n noeuds ?

La minoration de la hauteur est une conséquence directe du résultat de la question précédente :
De $n \leq 2^{h+1}-1$ on déduit $n+1 \leq 2^{h+1}$. La fonction $\log_2$ étant strictement croissante, on obtient :
$\log_2(n+1) \leq h+1$. La minoration est donc : $h \geq \log_2(n+1)-1$

5. Montrez que le nombre de branches vides --nombre de fils gauches et de fils droits vides-- d'un arbre à n noeuds est égal à n+1.
Indication : on distinguera les noeuds ayant zéro, un et deux fils.

Démonstration par récurrence :
Soit bv(t) le nombre de branches vides d'un arbre binaire t.
Pour n=1 : un arbre binaire t à un nœud se réduit à sa racine qui est aussi feuille. Le nombre de branches vides de t est bv(t)=2 et n+1=2.
Hypothèse de récurrence : On suppose que l'égalité bv(t)=n+1 reste vérifiée jusqu'à un certain k≥1 : pour tout arbre t à n≤k noeuds, bv(t)=n+1.
Considérons maintenant un arbre t à k+1 nœuds : Soit t' un arbre obtenu de t par suppression d'une feuille. t' a k nœuds et, d'après l'hypothèse de récurrence, k+1 branches vides. En remettant la feuille enlevée, on ré-obtient t, dont le nombre de branches vides est celui de t' plus 1 : bv(t)=bv(t')+1=(k+1)+1

Conclusion : pour tout arbre binaire t à n nœuds, bv(t)=n+1

6. Montrez que le nombre de feuilles est inférieur ou égal à $\frac{n+1}{2}$ et qu'il y a égalité si et seulement si chaque noeud de l'arbre est soit une feuille, soit a deux fils.

Indication : se servir du raisonnement utilisé à la question précédente.

Ce qu'il faut montrer est donc :

- f≤$\frac{n+1}{2}$
- f=$\frac{n+1}{2}$ si et seulement si l'arbre n'a pas de nœud de degré 1

Démonstration par l'absurde : Supposons que f>$\frac{n+1}{2}$. Comme chaque feuille donne naissance à deux branches vides, le nombre de telles branches serait strictement supérieur à n+1, ce qui serait en contradiction avec la réponse à la question précédente. Donc f≤$\frac{n+1}{2}$.

Si f=$\frac{n+1}{2}$, le nombre de branches vides issues des feuilles est n+1, c'est-à-dire le nombre maximal de telles branches que l'arbre peut avoir. Donc l'arbre n'a pas de nœud de degré 1 car un tel nœud ferait passer le nombre de branches vides à n+2.

Inversement, supposons que l'arbre n'a pas de nœud de degré 1. Donc toutes les branches vides sont issues des feuilles : bv(t)=2*f. D'après la réponse à la question précédente, bv(t)=n+1. Il s'ensuit que 2*f=n+1 et donc f=$\frac{n+1}{2}$.

7. Montrez que le nombre de feuilles d'un arbre est égal au nombre de noeuds de degré deux, plus un.
Soit t' l'arbre obtenu de t par ajout du fils manquant à chacun des nœuds de degré 1. Si f' et n' sont le nombre de feuilles et le nombre de nœuds, respectivement, de t', on a : f'=f+n_1 et n'= $n + n_1$. Et comme t' n'a pas de nœud de degré 1, d'après la réponse à la question précédente, on a : f'=$\frac{n'+1}{2}$=$\frac{n+n_1+1}{2}$. Donc f+n_1=$\frac{n+n_1+1}{2}$. D'où 2f+2n_1= $n + n_1 + 1$. Il s'ensuit que f=n-n_1-f+1. Or n-n_1-f =n_2

Conclusion : f=n_2+1

Complexité du tri par comparaison

1. Quel est le nombre de feuilles d'un tel arbre de décision ?
   Un arbre de décision pour le tri par comparaison de n éléments a autant de feuilles qu'il y a de permutations possibles de n éléments, c'est-à-dire n!

2. En déduire une borne inférieure sur la hauteur de l'arbre de décision.
   On a vu précédemment que f≤2^h, ce qui équivaut, grâce à la monotonie de la fonction $\log_2$, à h≥$\log_2 f$ avec ici f=n!. Donc h≥$\log_2(n!)$

3. En déduire une borne inférieure sur la complexité du tri par comparaion de n éléments.
Indication : d'après la formule de Stirling, on a n!>$(\frac{n}{e})^n$
La longueur d'un chemin de la racine à une feuille dans un arbre de décision est égale au nombre de comparaisons nécessaires au tri pour parvenir à la réponse souhaitée. La longueur du plus long chemin de la racine à une feuille -qui est égale par définition à la hauteur de l'arbre- nous donne donc la complexité T(n) de l'algorithme dans le pire cas. D'après la question précédente, la hauteur d'un

tel arbre de décision, quel que soit l'algorithme de tri par comparaison, est au moins de $\log_2(n!)$. Donc, en utilisant la formule de Stirling : T(n)≥ $\log_2(n!)$>n$\log_2 \frac{n}{e}$. Comme $\log_2(ab)$=$\log_2 a$+$\log_2 b$: T(n)=Ω(n$\log_2 n$)

Arbres binaires de recherche

1. Montrez que le temps de création d'un arbre binaire de recherche à partir d'une liste quelconque de n éléments est Ω(n$\log_2 n$).

Partant d'une liste quelconque de n éléments, si nous construisons un arbre binaire de recherche que nous parcourons en affichant les clés suivant un parcours (en profondeur d'abord) infixe, on obtient la liste des n éléments triés. Vu le résultat obtenu à l'exercice précédent la complexité de cette manipulation, qui est un tri par comparaison, est Ω(n$\log_2 n$). Le parcours avec affichage étant de complexité O(n), la construction de l'arbre binaire de recherche est Ω(n$\log_2 n$).

2. _Ecrivez un algorithme qui teste si un arbre binaire est un arbre binaire de recherche.

L'algorithme ci-dessous est basé sur la propriété suivante : un arbre binaire est de recherche si et seulement si son parcours (en profondeur d'abord) infixe permet l'affichage des clés des différents nœuds dans l'ordre croissant. L'algorithme est écrit sous forme d'une fonction booléenne ArbreBinRech®, dont l'argument r est un pointeur sur la racine d'un arbre binaire, et retournant VRAI si et seulement si l'arbre binaire dont la racine est pointée par r est un arbre binaire de recherche.

```
ArbreBinRech(r){
        Si r≠NIL{
                Si ArbreBinRech(r->sag){
                        Compteur++ ;
                        Si compteur=1{
                                Précédent=r->clé ;
                                ArbreBinRech(r->sad) ;
                                }
                        Sinon
                                Si précédent>r->clé alors retourner FAUX
                                Sinon{
                                        Précédent=r->clé ;
                                        ArbreBinRech(r->sad) ;
                                        }
                        }
                Sinon retourner FAUX ;
                }
        Sinon retourner VRAI ;    /* l'arbre vide est un arbre binaire de recherche */
        }
main(){
        …
        Compteur=0 ;
        ArbreBinRech(u0) ;
        …
        }
```