Université Paris Diderot – Paris 7
L3 Informatique

Algorithmique
Année 2008-2009, 1^{er} semestre

TD n°5

Résolution de récurrences

1 AVL

La première partie du TD consiste à revenir sur le TD 4 si nécessaire.

2 Récurrence et complexité

Lorsque l'on analyse la complexité d'un algorithme récursif sur un arbre binaire, on dispose dans le meilleur des cas d'une relation de récurrence de la forme $T(n) = 2T(n/2) + \Theta(f(n))$. Nous allons résoudre cette équation de récurrence dans quatre cas classiques. Dans ce qui suit, a et b désignent des constantes positives.

1. Calculer la complexité d'un algorithme dont la relation de récurrence est :

$$T(n) = \begin{cases} a \text{ si } n = 0 \\ 2T(n/2) + b \text{ si } n > 0 \end{cases}$$

2. Calculer la complexité d'un algorithme dont la relation de récurrence est :

$$T(n) = \begin{cases} a \text{ si } n = 0 \\ 2T(n/2) + b\log(n) \text{ si } n > 0 \end{cases}$$

3. Calculer la complexité d'un algorithme dont la relation de récurrence est :

$$T(n) = \begin{cases} a \text{ si } n = 0 \\ 2T(n/2) + bn \text{ si } n > 0 \end{cases}$$

4. Calculer la complexité d'un algorithme dont la relation de récurrence est :

$$T(n) = \begin{cases} a \text{ si } n = 0 \\ 2T(n/2) + bn^2 \text{ si } n > 0 \end{cases}$$

Remarque : pour mémoire, rappelons enfin la forme générale de la solution des équations de récurrence

$$T(n) = \alpha T(\frac{n}{\beta}) + f(n)$$

selon la fonction (croissante) $f(n)$. Dans ce qui suit, $\log_\beta \alpha$ dénote le logarithme de α en base β.

1. Si $f(n) = O(n^{\log_\beta \alpha - \epsilon})$ pour un certain $\epsilon > 0$, alors $T(n) = \Theta(n^{\log_\beta \alpha})$.
2. Si $f(n) = \Theta(n^{\log_\beta \alpha})$, alors $T(n) = \Theta(n^{\log_\beta \alpha} \log n)$.
3. Si $f(n) = \Omega(n^{\log_\beta \alpha + \epsilon})$ pour un certain $\epsilon > 0$, alors $T(n) = \Theta(f(n))$.

On trouvera plus de détails à ce sujet (dont la démonstration de ce résultat) dans le livre de Cormen, Leiserson, Rivest, Stein, *Introduction to Algorithms.*