TD n°5

Résolution de récurrences

1 AVL

La premiere partie du TD consiste a revenir sur le TD 4 si necessaire.

2 Récurrence et complexité

Lorsque l'on analyse la complexite d'un algorithme recursif sur un arbre binaire, on dispose dans le meilleur des cas d'une relation de recurrence de la forme $T(n) = 2T(n/2) + \quad (f(n))$. Nous allons resoudre cette equation de recurrence dans quatre cas classiques. Dans ce qui suit, a et b designent des constantes positives.

1. Calculer la complexite d'un algorithme dont la relation de recurrence est :
$$T(n) = \begin{cases} a \text{ si } n = 0 \\ 2T(n/2) + b \text{ si } n > 0 \end{cases}$$
2. Calculer la complexite d'un algorithme dont la relation de recurrence 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 complexite d'un algorithme dont la relation de recurrence est :
$$T(n) = \begin{cases} a \text{ si } n = 0 \\ 2T(n/2) + bn \text{ si } n > 0 \end{cases}$$
4. Calculer la complexite d'un algorithme dont la relation de recurrence est :
$$T(n) = \begin{cases} a \text{ si } n = 0 \\ 2T(n/2) + bn^2 \text{ si } n > 0 \end{cases}$$

Remarque : pour memoire, rappelons en n la forme generale de la solution des equations de recurrence

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

selon la fonction (croissante) $f(n)$. Dans ce qui suit, log α denote le logarithme de α en base β.

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

On trouvera plus de details a ce sujet (dont la demonstration de ce resultat) dans le livre de Cormen, Leiserson, Rivest, Stein, *Introduction to Algorithms*.