Analyse des Algorithmes Récursifs

Un Algorithme (fonction ou procédure) est dit Récursif si il fait appel à lui même.

l'implémentation récursive suit toujours la définition récursive de la solution, et nécessite la condition d'arrêt de la récursivité

exemple

$$n! = \begin{cases} n * (n-1)! \\ 1 \quad \text{si } n = 0 \end{cases}$$

⇒ Algorithme Fact (n : entier) : entier

Début

si n = 0 alors return 1 ← condition d'arrêt de la récursivité

sinon return n * Fact(n-1) ← Appel récursif

fsi;

fin;

Comment Analyser la complexité temporelle ?

1- déterminer l'equation de récurrence correspondante à T(n)

2- Resoudre cette equation de recurance

réc ②

il existe trois méthodes de résolution

a) Par délatation : par substitution de proche en proche jusqu'à trouver le cas général et terminer (sortir) à l'aide de la Condition d'arrêt

b) Deviner une solution, puis la démontrer par récurrence.

c) Utiliser la solution de certaines equations de récurances connues en mathématique.

Exemple de factoriel

1) equation de récurance

$$T_{fact}(n) = \begin{cases} T_{fact}(n-1) + 4 & \text{si } n > 0 \\ \text{si } n = 0 : 2 \end{cases}$$

2) résolution par délatation

$$T(n) = T(n-1) + 4$$
$$= (T(n-2)+4) + 4 = T(n-2) + 2\times 4$$
$$= (T(n-3)+4) + 2\times 4$$
$$= T(n-3) + 3\times 4$$
$$\vdots$$
$$= T(n-n) + n\times 4$$
$$= T(0) + n\times 4 = 4n + 2$$

$$T(n) = 4n + 2 = O(n)$$

fonction fact (n : entier) : entier

```
début
  si n = 0
  alors return 1
  sinon F := n * Fact(n-1)
        return F;
fsi;
fin.
```

réc (2)