TRI FUSION (« MERGE SORT »)

INTRODUCTION

Diviser pour régner

C'est algorithme est une illustration du principe algorithmique « diviser pour régner ».

La résolution se fait en trois étapes :

1. division du problème en un certain nombre de sous-problèmes ;
2. résolution des sous-problèmes récursivement ;
3. assemblage des solutions des sous-problèmes.

Dans le tri fusion

Si A est un tableau de longueur n.

1. On divise A en deux tableaux A_1 et A_2 de longueur divisée par deux :
2. on trie A_1 et A_2 ;
3. on fusionne les deux tableaux A_1, A_2 triés.

1. FUSION DE DEUX TABLEAUX TRIÉS

1.1. Principe

On se donne deux tableaux triés A, B. L'objectif est d'obtenir C tel que $|C| = |A|+|B|$ et tel que C soit la fusion triée de A et B.

Exemple. — Si $A = (1, 2, 3, 7, 9, 11)$ et $B = (2, 4, 5, 13)$ alors $C = (1, 2, 2, 3, 4, 5, 7, 9, 11, 13)$.

Principe. — Si C est indicé par k allant de 0 à $n + m - 1$ (avec $n = |A|, m = |B|$), alors :

1. on maintient i et j positions respectives dans A et B et on compare $A[i]$ et $B[j]$;
2. on insère le plus petit des deux dans la liste C et on augmente le compteur concerné de 1 ;
3. cas spécial quand l'un des deux tableaux est déjà trié (auquel cas aucune comparaison n'est nécessaire, on adjoint simplement).

```
function Merge(A,B)
        n = longueur(A), m = longueur(B)
        i = 0, j = 0
        for k from 0 to n+m-1 do
                if i > n-1 then
                        C[k] = B[j], j = j+1
                        continue
                if j > m-1 then
                        C[k] = A[i], i = i+1
                        continue
                if A[i] > B[i] then
                        C[k] = B[j], j = j+1
                else
                        C[k] = A[i], i = i+1
        return C
```

1.2. Preuve de correction

Proposition 1.2.0.1 (Invariant). —
Soit $[[k]], [[i]], [[j]]$ les valeurs respectives des variables k, i, j avec $[[k]] = [[i]] + [[j]]$. À l'issue d'une itération de la boucle *for*, les indices de 0 à $[[k]] - 1$ de C contiennent les éléments de :
- A entre 0 et $[[i]] - 1$,
- B entre0 et $[[j]] - 1$,

et triés dans l'ordre croissant.

Démonstration 1.2.0.1. —
On procède par récurrence sur k :
- Pour $[[k]] = 1$ c'est clair.
- Si $[[k]] - 1$ indices sont vérifiés, alors $[[k]]$-ième est :

$$C[k] = \min\{A[i], B[j]\} = \min\{A[h] : i \leq h \leq n - 1, B[k] : j \leq k \leq n - 1\}.$$

La propriété est bien vérifiée.

1.3. Complexité

On évalue le nombre de comparaisons.

— On effectue au plus trois comparaisons par itération, si $C(n)$ est le nombre de comparaisons :

$$C(n) = \mathrm{O}(n+m),\ C(n) = \Omega(n+m).$$

2. ALGORITHME POUR LE TRI FUSION

2.1. Algorithme

```
function MergeSort(A)
        n = longueur(A)
        if n <= 0 then return A
        m = int((n-1)/2)
        A1 = A[0 : m-1]
        A2 = A[m : n-1]
        A1 = MergeSort(A1)
        A2 = MergeSort(A2)
        A = Merge(A1,A2)
        return A
```

Illustration. — Par exemple :

$$
\begin{aligned}
&\begin{pmatrix}7 & 3 & 4 & 8 & 2 & \mid & 15 & 5 & 6\end{pmatrix}\\
\to &\begin{pmatrix}7 & 3 & \mid & 4 & 8\end{pmatrix} \begin{pmatrix}2 & 15 & \mid & 5 & 6\end{pmatrix}\\
\to &\begin{pmatrix}7 & 3\end{pmatrix} \begin{pmatrix}4 & 8\end{pmatrix} \begin{pmatrix}2 & 15\end{pmatrix} \begin{pmatrix}5 & 6\end{pmatrix}\\
\to &\begin{pmatrix}7\end{pmatrix} \begin{pmatrix}3\end{pmatrix} \begin{pmatrix}4\end{pmatrix} \begin{pmatrix}8\end{pmatrix} \begin{pmatrix}2\end{pmatrix} \begin{pmatrix}15\end{pmatrix} \begin{pmatrix}5\end{pmatrix} \begin{pmatrix}6\end{pmatrix} \quad \text{(fin division)}\\
\to &\begin{pmatrix}3 & 7\end{pmatrix} \begin{pmatrix}4 & 8\end{pmatrix} \begin{pmatrix}2 & 15\end{pmatrix} \begin{pmatrix}5 & 6\end{pmatrix}\\
\to &\begin{pmatrix}3 & 4 & 7 & 8\end{pmatrix} \begin{pmatrix}2 & 5 & 6 & 15\end{pmatrix}\\
\to &\begin{pmatrix}2 & 3 & 4 & 5 & 6 & 7 & 8 & 15\end{pmatrix} \quad \text{(fin fusion)}
\end{aligned}
$$

2.2. Complexité

Soit $C(n)$ le nombre de comparaisons pour l'algorithme de tri fusion. Alors

$$C(n) = C(\lfloor n/2 \rfloor) + C(\lceil n/2 \rceil) + n - 1 \text{ [1§]}$$

1§. Avec un algorithme plus performant pour la fusion on peut avoir un coût de $n-1$.

THÉORÈME 2.2.0.1. —
Si A est un tableau de longueur n, alors le nombre de comparaisons, $C(n)$, effectuées par le tri fusion est de $\mathrm{O}(n \log n)$ dans le pire des cas.

C'est un cas particulier du « MASTER Theorem ».

DÉMONSTRATION 2.2.0.2. —
On cherche à majorer la suite (t_n) définie par $t_0 = t_1 = 0$ et

$$t_n \leq n - 1 + t_{\lfloor n/2 \rfloor} + t_{\lceil n/2 \rceil}.$$

On suppose d'abord que $n = 2^k$ pour un certain $k \in \mathbf{N}$. On étudie (u_k) telle que $u_0 = u_1 = 0$ et :

$$u_{k+1} = 2^{k+1} - 1 + 2u_k.$$

On peut montrer que le terme général de cette suite est :

$$u_k = (k-1)2^k + 1.$$

On a donc

$$t_{2^k} \leq u_k = (k-1)2^k + 1.$$

Soient $n \in \mathbf{N}^*$ et $k \in \mathbf{N}^*$ tels que

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

Montrons que $t_n \leq u_{k+1}$:
- pour $n = 2$ c'est vérifié, $2^0 < n \leq 2^1$ et $t_2 = u_1$;
- supposons que c'est montré pour $n+1$ tel que $2^k < n+1 \leq 2^{k+1}$, on a

  $$t_{n+1} \leq n + t_{\lfloor n/2 \rfloor} + t_{\lceil n/2 \rceil}$$

  or $\lceil (n+1)/2 \rceil \leq 2^k \leq n$ et donc

  $$t_{n+1} \leq 2^{k+1} - 1 + u_k + u_k$$

  et donc

  $$t_{n+1} \leq u_{k+1}.$$

Ainsi, si k est tel que $2^k < n \leq 2^{k+1}$ alors

$$t_n \leq u_{k+1} = k2^{k+1} + 1$$

c'est-à-dire :

$$t_n < (2n)\log_2(n) + 1.$$