Devoir en temps libre n° 2 — corrigé

I Quelques exemples de calculs de complexité

I.1 Un premier exemple

EXERCICE 1 (*Minimum et maximum d'une liste*)

1. Soit $d \in \mathcal{D}_n$ une liste de longueur $n \geqslant 1$. On entre dans le troisième cas du filtrage, pour lequel il y a deux appels récursifs à la queue de longueur $n-1$, notée q, et des opérations en temps constant. On a donc $T(d) = 2T(q) + \Theta(1)$ et donc $T(n) \leqslant 2T(n-1) + \Theta(1)$ seulement a priori (Le cas au pire pour d implique-t-il un cas au pire pour q ?). Il suffit de remarquer que $\sup_{d=t::q\in\mathcal{D}_n} T(q) = \sup_{q\in\mathcal{D}_{n-1}} T(q)$ pour conclure que $\boxed{T(n) = 2T(n-1) + \Theta(1).}$

2. $\displaystyle\sum_{k=1}^{n} \frac{1}{2^k} = 1 - \frac{1}{2^n} \leqslant 1$ donc $\boxed{\displaystyle\sum_{k=1}^{n} \frac{1}{2^k} = \Theta(1).}$

3. Pour $k \geqslant 1$ on a $T(k) - 2T(k-1) = \Theta(1)$ et on divise par 2^k : $\boxed{u_k - u_{k-1} = \Theta\left(\frac{1}{2^k}\right).}$

4. On a donc $u_n - u_0 = \displaystyle\sum_{k=1}^{n} u_k - u_{k-1} = \Theta(1)$ et donc $\boxed{T(n) = \Theta\left(2^n\right).}$

5. On obtient un arbre d'appels récursifs, complet, de profondeur 3. Chaque nœud interne possède deux fils identiques. Le nombre de nœuds est $2^4 - 1 = 15$. $\boxed{\text{On obtient une complexité exponentielle}}$ puisqu'à chaque étape le nombre d'appels récursif double.

6. Il est dommage de calculer deux fois exactement la même chose. Il suffit de mémoriser le résultat à l'aide d'une liaison locale :

OCAML
```
| tete :: queue ->
      let mini, maxi = min_et_max queue
      (min tete mini, max tete maxi)
```

L'équation de récurrence devient alors $T(n) = T(n-1) + \Theta(1)$ dont la solution est bien une complexité linéaire.

☞ **Remarque 1**

On peut aussi commencer par calculer le min *et le* max *d'une liste, indépendamment, en $\Theta(n)$, et faire ensuite un appel à ces deux fonctions. On obtient aussi une complexité linéaire en $\Theta(2n) = \Theta(n)$. Mais un seul parcours de liste est souvent exigé à CCP par exemple.*

I.2 Complexité du tri fusion

1. Cf. corrigé TP n° 6.

2. Pour une liste de longueur n, il y a $\lceil n/2 \rceil$ appels récursifs à la fonction `division`, de coûts constants, pour une complexité finale linéaire $\boxed{T_{\text{division}}(n) = \Theta(n).}$

   Pour la fonction `fusion`, on choisit comme taille n la somme des longueur des deux listes. En dehors des deux cas de base, cette taille diminue de 1 à chaque appel récursif et il y a donc au plus n appels récursifs pour une complexité linéaire en $O(n)$. Un pire cas possible est celui où la première liste est formée d'un élément plus grand que les $n-1$ éléments de la deuxième liste, ce qui montre que la complexité est en $\boxed{T_{\text{fusion}}(n) = \Theta(n).}$

3. Pour $n \geqslant 0$, notons $c_n = T_{\text{division}}(n) + T_{\text{fusion}}(n)$ le coût dans le pire cas cumulé de la division et de la recombinaison d'une entrée de taille n. On a bien $c_n \geqslant 0$, on a vu que $c_n = \Theta(n)$ et on a bien que $(T_{\text{division}}(n))_{n\geqslant 1}$ et $(T_{\text{fusion}}(n))_{n\geqslant 1}$ sont croissantes et donc $(c_n)_{n\geqslant 1}$ également. Pour une liste de taille $n \geqslant 2$, on entre dans le troisième cas du filtrage de la fonction `tri_fusion`, dont le coût au pire est celui de la division, de la fusion et [1] de deux appels récursifs sur deux

1. Plus un éventuel coût constant que l'on peut intégrer dans c_n.

listes de tailles $\lfloor \frac{n}{2} \rfloor$ et $\lceil \frac{n}{2} \rceil$. On a alors bien une équation de récurrence de la forme :

$$T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + c_n \tag{1}$$

avec $a = b = 1$.

4. Pas de question.

5. On a pour $j \in \mathbb{N}^*$, $T\left(2^j\right) = 2T\left(2^{j-1}\right) + \Theta\left(2^j\right)$ et donc, en divisant par 2^j, on obtient $v_j - v_{j-1} = \Theta(1)$, puis, en sommant de 1 à k, $v_k = \Theta(k)$.

6. On a donc $T(n) = 2^k v_k = \Theta\left(2^k k\right) = \Theta(n \log n)$.

7. Montrons par récurrence sur $n \in \mathbb{N}^*$ que la suite $(T(k))_{k \in [\![1,n]\!]}$ est croissante.

   Pour initialiser, avec $n = 1$, la suite comporte un seul terme et il n'y a rien à faire.
   Pour $n = 2$, $T(2) = 2T(1) + c_2 \geqslant T(1)$.
   Supposons donc le résultat vrai jusqu'à un rang $n - 1 \geqslant 2$ et montrons que la suite $(T(k))_{k \in [\![1,n]\!]}$ est croissante. Comme la suite $(T(k))_{k \in [\![1,n-1]\!]}$ est croissante, il suffit de montrer que $T(n) \geqslant T(n-1)$.
   Comme $n - 1 \geqslant 2$, on a

   $$n > \lceil n/2 \rceil \geqslant \lceil (n-1)/2 \rceil \geqslant 1 \text{ et } n > \lfloor n/2 \rfloor \geqslant \lfloor (n-1)/2 \rfloor \geqslant 1$$

   et donc, par hypothèse de récurrence,

   $$T(\lceil n/2 \rceil) \geqslant T(\lceil (n-1)/2 \rceil) \text{ et } T(\lfloor n/2 \rfloor) \geqslant T(\lfloor (n-1)/2 \rfloor).$$

   Comme de plus $c_n \geqslant c_{n-1}$, on a donc

   $$T(n) \geqslant T(\lceil (n-1)/2 \rceil) + T(\lfloor (n-1)/2 \rfloor + c_{n-1} = T(n-1)$$

   ce qui établit la récurrence.

8. Soit $n \in \mathbb{N}^*$, il existe $k \in \mathbb{N}$ tel que $2^k \leqslant n < 2^{k+1}$ (on prend $k = \lfloor \log_2 n \rfloor$). Par croissance de $(T(n))_{n \in \mathbb{N}^*}$, on a $T(2^k) \leqslant T(n) \leqslant T(2^{k+1})$, donc

   $$T(n) = \Omega(T(2^k)) = \Omega(n \log n)$$

   et

   $$T(n) = O(T(2^{k+1})) = O((n+1)\log(n+1)) = O(n \log n).$$

   Finalement, $T(n) = \Theta(n \log n)$.

9. Cette complexité est bien, bien meilleure. Par exemple, pour une liste de taille 10^6 sur un ordinateur à un milliard d'opérations par seconde on passe de 1/4 h (pour le tri par sélection ou insertion) à 10ms !

II Problème : représentation d'images par des arbres quaternaires

Question II.6 Ce qui est demandé ici est de formaliser dans le langage mathématique la définition **II.5**. Nous proposons de formaliser les quatre axiomes :

$$\mathrm{AX}_1(a) \equiv \forall n \in \mathcal{N}(a),\ (\mathcal{X}(n), \mathcal{Y}(n), \mathcal{T}(n)) \in (\mathbb{N}^*)^3$$

$$\mathrm{AX}_2(a) \equiv \forall d \in \mathcal{D}(a),\ \forall \mathrm{o} \in \{\mathrm{SO}, \mathrm{SE}, \mathrm{NO}, \mathrm{NE}\},\ \mathcal{T}(f_{\mathrm{o}}(d)) = \mathcal{T}(d)/2$$

$$\mathrm{AX}_3(a) \equiv \forall d \in \mathcal{D}(a),\ \begin{cases} \mathcal{X}(f_{\mathrm{SO}}(d)) = \mathcal{X}(d) \wedge \mathcal{Y}(f_{\mathrm{SO}}(d)) = \mathcal{Y}(d) \\ \mathcal{X}(f_{\mathrm{SE}}(d)) - \mathcal{T}(f_{\mathrm{SE}}(d)) = \mathcal{X}(d) \wedge \mathcal{Y}(f_{\mathrm{SE}}(d)) = \mathcal{Y}(d) \\ \mathcal{X}(f_{\mathrm{NO}}(d)) = \mathcal{X}(d) \wedge \mathcal{Y}(f_{\mathrm{NO}}(d)) - \mathcal{T}(f_{\mathrm{NO}}(d)) = \mathcal{Y}(d) \\ \mathcal{X}(f_{\mathrm{NE}}(d)) - \mathcal{T}(f_{\mathrm{NE}}(d)) = \mathcal{X}(d) \wedge \mathcal{Y}(f_{\mathrm{NE}}(d)) - \mathcal{T}(f_{\mathrm{NE}}(d)) = \mathcal{Y}(d) \end{cases}$$

$$\mathrm{AX}_4(a) \equiv \forall d \in \mathcal{D}(a),\ \exists o_1, o_2 \in \{\mathrm{SO}, \mathrm{SE}, \mathrm{NO}, \mathrm{NE}\},\ \exists b_1 \in \mathcal{B}(f_{o_1}), \exists b_2 \in \mathcal{B}(f_{o_2}),\ o_1 \neq o_2 \wedge \mathcal{C}(b_1) \neq \mathcal{C}(b_2)$$

On a alors

$$\mathrm{VAQ}(a) \equiv \mathrm{AX}_1(a) \wedge \mathrm{AX}_2(a) \wedge \mathrm{AX}_3(a) \wedge \mathrm{AX}_4(a).$$