TD Invariants n°2

Quelques invariants

Ce TD comporte plusieurs exercices indépendants. L'exercice I est très simple. Les exercices II et III sont plus complexes. L'exercice IV est bien plus complexe.

I. Redécouvrir l'exponentiation rapide.

L'objectif de cet exercice est de calculer efficacement a^n, pour $a \in \mathbb{Q}$ et $n \in \mathbb{N}$. On supposera que la multiplication de deux rationnels est une opération élémentaire.

Q1. On suppose n pair. Supposons pouvoir calculer $a^{n/2}$ avec un algorithme $\mathscr{A}_{n/2}$. Construire un algorithme $\mathscr{A}_n$ calculant efficacement a^n. On utilisera habilement $\mathscr{A}_{n/2}$.

RÉPONSE. On remarque que, lorsque n est pair, on peut calculer $r \overset{\text{def}}{=} a^{n/2}$, puis le passer au carré, on a donc $r^2 = \left(a^{n/2}\right)^2 = a^n$. En s'inspirant de cette idée, on construit l'algorithme $\mathscr{A}_n$ comme ci-dessous.

> $r \leftarrow \mathscr{A}_{n/2}(a)$
> **Retourner** $r \times r$

Q2. Soit n une puissance de deux. Trouver un algorithme calculant efficacement a^n.

RÉPONSE. On applique la question Q1 plusieurs fois en divisant n par 2, jusqu'au cas $n = 1$ (impair), où $a^n = a$. On construit l'algorithme $\textsc{Pow}(a, n)$ comme ci-dessous.

> **Entrée** $a \in \mathbb{Q}$ et $n \in \mathbb{N}$ puissance de 2
> **Sortie** le nombre a^n
>
> **Si** $n = 1$ **alors**
> > **Renvoyer** a
> **Sinon**
> > $r \leftarrow \textsc{Pow}(a, n/2)$
> > **Renvoyer** $r \times r$
> **Fin si**

Q3. Avec un O, déterminer le nombre d'opérations élémentaires de l'algorithme trouvé en Q2. On supposera le calcul de $n/2$ à partir de n possible en une opération élémentaire.

RÉPONSE. Notons $T(n)$ le nombre d'opérations élémentaires (*i.e.* les multiplications d'entiers) utilisées par l'algorithme de Q2 sur une entrée $(a, n) \in \mathbb{Q} \times \mathbb{N}$. Alors,

$$T(n) \leq T\left(\frac{n}{2}\right) + 1 \leq T\left(\frac{n}{4}\right) + 2 \leq T\left(\frac{n}{8}\right) + 3 \leq \cdots \leq T(1) + \log_2 n \leq \mathrm{O}(\log n)$$

(Une démonstration par récurrence sur k pour $n = 2^k$ serait plus propre, mais l'idée est visible clairement sans trop se soucier des détails de la récurrence.)

Q4. Soit $\textsc{Pow}(a, n)$ l'algorithme de la question Q2. Prouver sa correction par récurrence, pour tout entier $n \in \mathbb{N}$, qui est une puissance de deux.

RÉPONSE. Montrons que $\textsc{Pow}(a, 2^k) = a^{(2^k)}$, quel que soit le couple $(a, k) \in \mathbb{Q} \times \mathbb{N}$. On le montre par récurrence sur k.

Initialisation. Pour $k = 0$, on a $n = 2^k = 1$, et $a^1 = a$. Or, $\textsc{Pow}(a, 1) = a$. D'où l'initialisation.

Hérédité. Supposons que $\textsc{Pow}(a, 2^k) = a^{(2^k)}$. Alors,

$$\textsc{Pow}(a, 2^{k+1}) \underset{\text{Algo}}{=} \textsc{Pow}(a, 2^k) \times \textsc{Pow}(a, 2^k) \underset{\text{Hyp}}{=} a^{(2^k)} \times a^{(2^k)} = a^{(2^k)+(2^k)} = a^{(2^{k+1})},$$

d'où l'hérédité.

Par récurrence ceci montre le résultat attendu.

Q5. Construire un algorithme calculant a^n pour $n \in \mathbb{N}$ qui n'est pas *forcément* une puissance de deux.

RÉPONSE. On s'inspire des questions précédente.
- Dans le cas d'un entier pair, on peut calculer $r = a^{n/2}$ puis r^2.
- Dans le cas d'un entier impair, on peut calculer $r = a^{(n-1)/2}$ puis $a \times r^2$.

On construit donc l'algorithme suivant (nommé aussi $\textsc{Pow}$).

Entrée $a \in \mathbb{Q}$ et $n \in \mathbb{N}$
Sortie le nombre a^n

Si $n = 1$ **alors**
  Renvoyer a
Sinon si n est pair **alors**
  $r \leftarrow \textsc{Pow}(a, n/2)$
  Renvoyer $r \times r$
Sinon
  $r \leftarrow \textsc{Pow}(a, (n-1)/2)$
  Renvoyer $a \times r \times r$
Fin si

II. Tri en une seule boucle ?

Considérons l'algorithme ci-contre. La tableau A commence à 0.

Q1. Donner les étapes de l'algorithme sur le tableau A :

$$A = [10, 20, 30, 40, 50, 25]$$

jusqu'à la fin de l'algorithme. On s'intéressera particulièrement au contenu de A et la valeur de i après chaque itération de la boucle **Tant que**.

RÉPONSE. Voici les différentes valeurs que prend le tableau A. La valeur de $A[i]$ sera mise en gras pour suivre l'avancement de l'algorithme.

Entrée. Un tableau A de taille n.
Sortie. Le tableau A modifié.

$i \leftarrow 0$
Tant que $i < n$ **faire**
  Si $i = 0$ ou $A[i] \geq A[i-1]$ **alors**
    $i \leftarrow i + 1$
  Sinon
    Échanger $A[i]$ et $A[i-1]$
    $i \leftarrow i - 1$
  Fin si
Fin tant que

$$
\begin{array}{ll}
[\mathbf{10}, 20, 30, 40, 50, 25] & i = 0 \\
[10, \mathbf{20}, 30, 40, 50, 25] & i = 1 \\
[10, 20, \mathbf{30}, 40, 50, 25] & i = 2 \\
[10, 20, 30, \mathbf{40}, 50, 25] & i = 3 \\
[10, 20, 30, 40, \mathbf{50}, 25] & i = 4 \\
[10, 20, 30, 40, 50, \mathbf{25}] & i = 5 \\
[10, 20, 30, 40, \mathbf{25}, 50] & i = 4 \\
[10, 20, 30, \mathbf{25}, 40, 50] & i = 3 \\
[10, 20, \mathbf{25}, 30, 40, 50] & i = 2 \\
[10, 20, 25, \mathbf{30}, 40, 50] & i = 3 \\
[10, 20, 25, 30, \mathbf{40}, 50] & i = 4 \\
[10, 20, 25, 30, 40, \mathbf{50}] & i = 5 \\
[10, 20, 25, 30, 40, 50] & i = 6
\end{array}
$$

Q2. Expliquer pourquoi l'algorithme trie n'importe quel tableau A. Introduire un invariant $(\mathscr{I})$. Le démontrer. Conclure sur la correction de l'algorithme.

RÉPONSE. L'algorithme trie la liste de gauche à droite. De l'exemple précédent, on introduit l'invariant :

$$(\mathscr{I}) \qquad \forall k \in [\![0, i-2]\!], A[k] \leq A[k+1]$$

Démontrons que $(\mathscr{I})$ est bien un invariant.
- Pour l'initialisation, $i = 0$ et donc $[\![0, i-2]\!] = \emptyset$, d'où $(\mathscr{I})$ est vrai.
- Notons $\underline{i}$ et $\underline{A}$ les valeurs de i et de A avant la boucle, et $\overline{i}$ et $\overline{A}$ les valeurs de i et A après la boucle.

Q3. Démontrer que l'algorithme termine à l'aide d'un variant. Donner un majorant raisonnable du temps d'exécution à l'aide d'un O.

III. Tri bulle.

On considère l'algorithme ci-après, que l'on nommera TriBulle(A). On supposera que le tableau A commence à 1.

Q1. Trouver un invariant $\mathscr{I}(i)$ qui est vrai à la fin de la boucle **Pour** extérieure.

Q2. Démontrer que l'invariant $\mathscr{I}(i)$ est bien vérifié, quel que soit i.

Q3. En déduire sur la correction du tri bulle.

Indication. Afin de visualiser comment fonction l'algorithme TriBulle, on pourra essayer un exemple.

Si vous ne voyez pas l'idée de l'algorithme, vous pourrez regarder cette animation.

```
Pour i ∈ [1, n] faire
    Pour j ∈ [1, n − i] faire
        Si A[j] > A[j + 1] faire
            Échanger A[j] et A[j + 1]
        Fin si
    Fin pour
Fin pour
```

IV. Un problème de graphes.

On considère un graphe pondéré orienté acyclique $G = (V, E, w)$, où $V = [\![1, n]\!]$. L'objectif de cet exercice est de trouver la longueur du plus long chemin dans G.

On fixe top un *tri topologique* de V. Un *tri topologique* de V est une permutation de V telle que, pour tout couple d'entiers $(i, j) \in [\![1, n]\!]^2$, si $(\mathrm{top}_i, \mathrm{top}_j) \in E$, alors $i \leq j$.

On considère l'algorithme ci-contre.

Dans cet algorithme, on supposera que le maximum sur l'ensemble vide est 0.

$(L_1, \ldots, L_n) \leftarrow \underbrace{(0, \ldots, 0)}_{n}$

Pour $i \in [\![1, n]\!]$ **faire**

  $v \leftarrow \mathrm{top}_i$

  $L_v \leftarrow \max\{L_u + w(u, v) \mid u \in V\}$

Fin pour

Renvoyer $\max_{i \in [\![1,n]\!]} L_i$

On supposera que $w(u, v)$ est nul si $(u, v) \notin E$.

Q1. Montrer que l'algorithme vérifie l'invariant $\mathscr{I}(i)$: après i itérations de la boucle **Pour**, l'élément L_{top_i} contient la longueur du plus long chemin qui termine par top_i.

Q2. Quelle est la complexité de l'algorithme ? On exprimera la réponse sous la forme d'un Θ en terme de V et E. Justifier votre réponse.

Ces exercices sont extraits des cours d'informatique « Algorithmes et structures de données » de l'École polytechnique fédérale de Zurich.