SOLUTIONS SÉRIE 1

Les exercices dénotés par une étoile (*) sont les exercices les plus importants à faire pour s'assurer de bien maîtriser la matière du cours. Il est recommandé de bien les comprendre. Si vous avez des questions, profitez des séances de travaux dirigés ou du forum pour les poser.

***Question # 1**
Dans une base b, tout nombre entier (non négatif) n s'écrit :

$$n = \sum_{i=0}^{d-1} a_i b^i \quad \text{avec } a_i \in \{0, 1, \ldots, b-1\}$$

où d est la taille de l'entier mesuré en nombre de chiffres qu'il occupe lorsque l'on écrit ce nombre, en base b, sur une feuille de papier.

A) *Démontrez que $d = \lfloor \log_b(n) \rfloor + 1 = \lceil \log_b(n+1) \rceil$*
Solution :
En partant du fait que :

— *Le plus petit entier utilisant d digits (en base b) est $n = b^{d-1}$*
— *Le plus grand entier utilisant d digits (en base b) est $n = b^d - 1$*

on obtient que :

$$b^{d-1} \leq n \leq b^d - 1 \tag{1}$$

— *premier résultat :*

$$\begin{aligned} & b^{d-1} \leq n \leq b^d - 1 \\ \Rightarrow\ & b^{d-1} \leq n < b^d \\ \Rightarrow\ & d - 1 \leq \log_b(n) < d \\ \Rightarrow\ & \lfloor \log_b(n) \rfloor = d - 1 \\ \Rightarrow\ & d = \lfloor \log_b(n) \rfloor + 1 \end{aligned}$$

— *deuxième résultat :*

$$\begin{aligned} & b^{d-1} \leq n \leq b^d - 1 \\ \Rightarrow\ & b^{d-1} < n + 1 \leq b^d \\ \Rightarrow\ & d - 1 < \log_b(n+1) \leq d \\ \Rightarrow\ & \lceil \log_b(n+1) \rceil = d \\ \Rightarrow\ & d = \lceil \log_b(n+1) \rceil \end{aligned}$$

— *en combinant ces deux résultats on a :*

$$d = \lfloor \log_b(n) \rfloor + 1 = \lceil \log_b(n+1) \rceil$$

B) *Quel est l'effet du choix de la base b sur la taille d de l'entier n ?*
Solution :
Plus la base b est grande, plus le nombre de digit d requis sera petit et vice versa. De plus, l'effet

d'un changement de base aura un impact faible sur le nombre de digit car $log_a n = log_a b \times log_b n$ *et* $log_a b$ *n'augmente pas rapidement en fonction de l'écart entre les différentes bases* $(b - a)$*, pour* $b > a > 1$.

***Question # 2**

A) Que fait l'algorithme d'Euclid pour une paire de nombre où le premier est plus petit que le second ?

Solution :

L'algorithme d'Euclid permute les nombres à la première itération car $a \mod b = a$ *lorsque* $a < b$.

B) Combien de fois une telle situation peut se produire pendant l'exécution de l'algorithme ?

Solution :

Une telle situation peut arriver au plus une fois (et seulement à la première itération) car $a \mod b$ *ne peut jamais être plus grand que* b*. Soit* (m_1, n_1) *la paire initiale, la paire après une itération sera* (m_2, n_2) *où* $m_2 = n_1$ *et* $n_2 = m_1 \mod n_1$*. Comme* n_2 *n'est pas plus grand que* n_1*, il ne peut être plus grand que* m_2.

***Question # 3**

Décrivez l'algorithme standard pour trouver la représentation binaire d'un entier positif :

A) En français.

Solution :

Diviser le nombre n *donné par 2 : le reste (0 ou 1) devient le prochain digit (de la droite vers la gauche) de la representation binaire. Remplacer* n *par le quotien de la dernière division. Répéter cette opération jusqu'à ce que* $n = 0$.

B) En pseudocode.

Solution :

Algorithm *Binaire*(n) :
  $k \leftarrow 0$
  WHILE $n \neq 0$
    $b_k \leftarrow n \mod 2$
    $n \leftarrow \lfloor n/2 \rfloor$
    $k \leftarrow k + 1$
  RETURN $[b_{k-1}, \ldots, b_0]$

***Question # 4**

Considérez l'algorithme suivant pour trouver l'écart entre les deux éléments les plus proches dans un tableau de nombres.

```
Algorithm MinDistance(A[0...n − 1]) :
    dmin ← ∞
    FOR i ← 0 TO n − 1
        FOR j ← 0 TO n − 1
            IF i ≠ j AND |A[i] − A[j]| < dmin
                dmin ← |A[i] − A[j]|
    RETURN dmin
```

Faites des modifications à cet algorithme afin de le rendre plus performant (Vous pouvez aussi refaire complètement l'algorithme selon une autre idée).

Solution :

On peut modifier l'algorithme ci-haut de la façon suivante. Ceci nous évite de recalculer plusieurs fois la même valeur.

```
Algorithm MinDistance2(A[0...n − 1]) :
    dmin ← ∞
    FOR i ← 0 TO n − 2
        FOR j ← i + 1 TO n − 1
            IF |A[j] − A[i]| < dmin
                dmin ← |A[j] − A[i]|
    RETURN dmin
```

On peut générer le nouvel algorithme suivant.

```
Algorithm MinDistance3(A[0...n − 1]) :
    trier(A)
    dmin ← ∞
    FOR i = 0 TO n − 2
        IF A[i + 1] − A[i] < dmin
            dmin ← A[i + 1] − A[i]
    RETURN dmin
```

Nous verrons plus tard que si l'on choisit judicieusement la façon de trier, l'algorithme MinDistance3 est le meilleur des trois.