TD de Algorithmique n° 1

6 octobre 2011
(Correction)

I) Représentation d'entiers en base b

On a appris à l'école à écrire un entier comme une séquence $x_n \cdots x_0$ de symboles dans un alphabet $A = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Ce qu'on fait, en vérité, est d'associer à chaque chiffre une valeur différente qui dépend de sa position (notation positionnelle) en supposant de travailler en base 10. Donc, on calcul la valeur de 1456 comme $1 \cdot 10^3 + 4 \cdot 10^2 + 5 \cdot 10^1 + 6 \cdot 10^0$.

De manière générale, donnée une base b, l'alphabet associé a b est $A_b = \{0, \ldots, b-1\}$ et la valeur associée au nombre $x_n \cdots x_0$ écrit en base b est déterminée par $\sum_{i=0}^{n} x_i \cdot b^i$.

Exercice 1 :

1. Les nombres suivants sont écrits en base 2 ; écrivez les en base 10 : 1011, 110011, 101010, 11100001.
2. Les nombres suivants sont écrits en base 10 ; écrivez les en base 2 : 5, 10, 123, 17, 1010.

Correction :

1. $(1011)_{2\to 10} = 2^3 \cdot 1 + 2^2 \cdot 0 + 2^1 \cdot 1 + 2^0 \cdot 1 = 8 + 2 + 1 = 11$.
   $(110011)_{2\to 10} = 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 32 + 16 + 2 + 1 = 51$.
   $(101010)_{2\to 10} = 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 32 + 8 + 2 = 42$.
   $(11100001)_{2\to 10} = 128 + 64 + 32 + 1 = 225$.
2. $(5)_{10\to 2} = 101$.
   $(10)_{10\to 2} = 1010$.
   $(123)_{10\to 2} = 1111011$.
   $(17)_{10\to 2} = 10001$.
   $(1010)_{10\to 2} = 1111110010$.

II) Tri par insertion

Il s'agit ici de comprendre et d'écrire la fonction qui trie par insertion. Le principe du tri par insertion consiste à considérer une à une les valeurs du tableau, et à les inserer au bon endroit dans le tableau constitué des valeurs précédemment considérées et triées. La partie déjà triée du tableau est donc parcourue de droite à gauche. Les éléments sont donc décalés vers la droite tant que l'élément à insérer est plus petit qu'eux. Dès que l'élément à insérer est plus grand qu'un des éléments du tableau trié, il n'y a plus de décalage et on insère l'élément dans la case laissée vacante.

Exercice 2 :

1. Simuler l'exécution du tri par insertion sur le tableau suivant :

$$A = \langle 78, 24, 17, 51, 1 \rangle$$

2. Ecrivez le pseudocode [1] de la fonction TRI_INSERTION(A), qui trie sur place, par insertion.

Correction :

1. Au debut le tableau se presente comme suive : $\langle 78, 24, 17, 51, 1\rangle$.
   Etape 1) On commence par inserer 24 dans le sous-tableau bleu qui est déjà trié : $\langle 24, 78, 17, 51, 1\rangle$.
   Etape 2) On insère 17 dans le sous-tableau bleu qui est déjà trié : $\langle 17, 24, 78, 51, 1\rangle$.
   Etape 3) On insère 51 dans le sous-tableau bleu qui est déjà trié : $\langle 17, 24, 51, 78, 1\rangle$.
   Etape 4) Finalement, on insère 1 et on obtient : $\langle 1, 17, 24, 51, 78\rangle$.
2. Voila le code :

```
TRI-INSERTION(A)
1  for j ← 2 to LENGTH(A)
2       do cle ← A[j]
3          i ← j − 1
4          while (i > 0) and (A[i] > cle)
5               do A[i + 1] ← A[i]
6                  i ← i − 1
7          A[i + 1] ← cle
```

III) Recherche binaire

On va décrire maintenant un algorithme de recherche binaire pour trouver un élément x dans un tableau trié $A[1, \ldots, n]$. L'idée de cet algorithme est de partager le tableau $A[1, \ldots, n]$ en deux parties qui ont, à peu prés, le même nombre d'éléments (c'est à dire $\frac{n}{2}$). Si $x = A[\lfloor \frac{n}{2} \rfloor]$, alors on l'a trouvé, si $x < A[\lfloor \frac{n}{2} \rfloor]$, alors on continue à le chercher dans $A[1, \ldots, \lfloor \frac{n}{2} \rfloor]$ sinon on continue notre recherche dans $A[\lfloor \frac{n}{2} \rfloor + 1, \ldots, n]$.

Exercice 3 :

1. Simuler l'ex'ecution de la recherche binaire de l'element 53 dans le tableau suivant :

$$A = \langle 11, 19, 27, 51, 53, 65, 99, 129, 190\rangle.$$

2. Ecrire une fonction (itérative) RECHERCHE-BIN(A, x) qui implémente l'algorithme de recherche décrit ci-dessus. RECHERCHE-BIN(A, x) donnera comme sortie i si $x = A[i]$, et -1 si l'élément x n'est pas présent dans le tableau A.
3. Pour quelle raison cet algorithme marche ? Donner l'idée intuitive de l'invariante de boucle qui fait que l'algorithme est correct. Où est ce qu'on utilise le fait que A est trié ?
4. (facultatif. Exprimer le coût de RECHERCHE-BIN(A, x) en fonction de la longueur n de A.)

Correction :

1. Etape 1) On divise en deux le tableau : $\langle 11, 19, 27, 51, 53, 65, 99, 129, 190\rangle$. Dès que $53 \neq 51$ on doit continuer à le chercher soit sur la moitié bleu soit sur la partie verte. On contrôle alors que $53 \not< 51$ et on le cherche dans la moitié verte.
   Etape 2) On divise en deux le tableau vert : $\langle 53, 65, 99, 129, 190\rangle$. Dès que $53 \neq 65$ on doit continuer à le chercher soit sur la moitié bleu soit sur la partie verte. On contrôle alors que $53 < 65$ et on le cherche dans la moitié bleu.
   Etape 3) On divise en deux le tableau bleu : $\langle 53, 65\rangle$. Dès que $53 \neq 65$ on doit continuer à le chercher soit sur la moitié bleu soit sur la partie verte. On contrôle alors que $53 = 53$ et on retour 5 comme sortie.

1. Utiliser les conventions qui sont décrites dans le livre de Cormen et Al. [CLRS01].

2. La fonction suivante implemente l'algorithme de recherche binaire :

```
RECHERCHE-BIN(A, x)
 1  i ← 1
 2  j ← LENGTH(A)
 3  med ← ⌊(i + j)/2⌋
 4  while (x ≠ A[med] and (i < j))
 5        do if (x < A[med])
 6              then j ← med
 7              else  i ← med + 1
 8           med ← ⌊(i + j)/2⌋
 9  if (x = A[med])
10     then return med
11     else  return −1
```

3. L'idée est que la cardinalité de l'ensemble $\{A[i], \ldots, A[j]\}$ qui peut contenir l'élément x qu'on cherche, devient la moitié à chaque itération. S'il n'y a pas d'occurrences de x dans le tableau A, quand même la quantité $j - i$ décroit. Donc, la boucle *while* termine toujours, soit parce que $i \not< j$ soit parce qu'on a trouvé x.
4. $O(\log n)$

Références

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. *Introduction to Algorithms*, second edition. The MIT Press/McGraw-Hill, 2001.