TP 4 : algorithmes au programme officiel

Judicaël Courant

28 novembre 2016

1 Présentation

Ce TP regroupe l'essentiel des algorithmes du programme officiel (à l'exception des algorithmes de calcul numérique). Vous êtes invités à laisser ceux que vous maîtrisez pour vous concentrer sur les autres.

2 Nombres

Étant donné un entier $B \geq 2$, on dira qu'un tableau t de taille n représente un entier naturel p en base B si t contient des entiers $c_0, \ldots, c_{n-1}$ vérifiant $c_i \in [\![0, B[\![$ et $p = \sum_{i=0}^{n-1} c_i B^i$.

Par exemple, en base 5, le tableau $[3, 2, 1, 0]$ représente le nombre 38. Notez que la représentation d'un nombre n'est pas unique puisque 38 peut aussi être représenté par $[3, 2, 1]$ et $[3, 2, 1, 0, 0]$.

Q0 Écrire une fonction `nombre(t, B)` qui, à partir de la représentation d'un nombre donnée dans t, retourne le nombre représenté par t en base B.

Vous testerez votre fonction avec les représentation de 38 en base 5 données plus haut.

Q1 Écrire une fonction `representation(p, B)` qui retourne une représentation en base B de l'entier p. Plus précisément, on retournera le tableau le plus court, sauf si p est nul, auquel cas on retournera le tableau $[0]$.

Vous testerez votre fonction pour représenter l'entier 38 en base 5 ainsi que pour l'entier 28 en base 5.

3 Maximum

Q2 Écrire une fonction `maxi` retournant le maximum d'un tableau de nombres non-vide.

Q3 Écrire une fonction `imaxi` retournant l'indice d'un maximum d'un tableau de nombres non-vide. En fera en sorte de ne faire qu'un seul passage sur le tableau et on écrira cette fonction de la façon la plus simple possible.

Vous testerez vos fonctions avec les tableaux $[10]$, $[23, 42, 5, 6]$, $[6, 5, 34, 42]$, $[23, 24, 37, 12, 37, 15]$.

4 Recherche dans un tableau

4.1 Recherche séquentielle

Q4 Écrire une fonction `indice(t, x)` retournant un indice où x apparaît dans le tableau t s'il apparaît et -1 s'il n'apparaît pas.

Vous essayerez votre fonction avec la chaîne `"toto"` pour x et pour t vous prendrez successivement `["toto", "titi", "tata", "tutu"]`, `["titi", "tata", "tutu", "toto"]`, `["titi", "toto", "tata", "tutu"]`, `["titi", "tata", "tutu"]`.

4.2 Recherche par dichotomie dans un tableau trié

Q5 Écrire une fonction `dicho1(t, x)` retournant un indice où x apparaît dans le tableau t s'il apparaît et -1 s'il n'apparaît pas. À la différence de la question précédente, on supposera que le tableau t est trié par ordre croissant. Vous effectuerez donc une recherche par dichotomie. Pour cette question, vous écrirez votre fonction de façon itérative, en précisant soigneusement l'invariant de boucle que vous utilisez.

Vous essayerez votre fonction avec le tableau `["tata", "tete", "titi", "toto", "tutu", "tyty"]` dans lequel vous chercherez successivement `"tata"`, `"toto"`, `"tyty"`, `"abc"`, `"tatou"`, `"tyran"` et `"ut"`.

Q6 Écrire une fonction `dicho2(t, x)` respectant les mêmes spécifications que `dicho1(t, x)` mais cette fois-ci de façon récursive : au lieu d'utiliser une boucle, vous écrirez une fonction auxiliaire récursive que vous utiliserez.

Vous essayerez votre fonction avec les mêmes tests que pour `dicho1(t, x)`.

4.3 Recherche d'une sous-chaîne

Q7 Écrire une fonction `cherche(u, s)` qui cherche un indice auquel la sous-chaîne u apparaît dans la chaîne de caractères s si elle apparaît et -1 si elle n'apparaît pas. Par exemple dans la chaîne `"abracadabra"`, la sous-chaîne `"raca"` apparaît à l'indice 2. En revanche la sous-chaîne `"racb"` n'apparaît pas.

Vous testerez votre fonction avec la chaîne `"abracadabra"` et les sous-chaînes `""` (chaîne vide), `"abrac"`, `"dabra"`, `"rade"`.

5 Tris

5.1 Tri par insertion

Q8 Écrire une fonction `tri_insertion(t)` prenant en argument un tableau t et le triant par ordre croissant. On aura intérêt à écrire une fonction auxiliaire `insere(t, i)` prenant en argument un tableau t dont les i premiers éléments sont triés et insérant le i^{e} à sa place.

Vous essaierez cette fonction avec les tableaux suivants : `[]` (le tableau vide), `[42]`, `[17, 42]`, `[42, 17]`, `[5, 17, 42]`, `[5, 42, 17]`, `[17, 5, 42]`, `[17, 42, 5]`, `[42, 5, 17]`, `[42, 17, 5]`, `[i for i in range(10000)]`, `[randrange(1000000) for i in range(10000)]`

(la fonction `randrange` vient du module `random`), `[-i for i in range(10000)]`. (Vous penserez notamment à comparer le temps de calcul pour les trois derniers tableaux.)

Q9 Écrire une fonction `fusion(t1, t2)` prenant en argument deux tableaux t_1 et t_2 supposés triés, de tailles respectives n_1 et n_2 et retournant un tableau t trié, de taille $n_1 + n_2$ et contenant les mêmes éléments que t_1 et t_2 (avec mêmes multiplicités). On écrira cette fonction itérativement avec une boucle définie (boucle `for`).

Q10 Écrire une fonction `tri_fusion(t)` prenant en argument un tableau t et retournant un nouveau tableau, trié et contenant les mêmes éléments que t avec mêmes multiplicités. Le principe est le suivant : si trier t n'est pas trivial, on partage t en deux tableaux t_1 et t_2 de même taille environ, qu'on trie récursivement, ce qui donne des tableaux u_1 et u_2. On fusionne alors ces deux tableaux, pour obtenir un tableau u qu'il suffit de retourner.

Vous testerez votre fonction sur les mêmes tableaux que pour le tri par insertion.

5.2 Tri pivot

Nous allons écrire une version du tri pivot avec recopie (le tableau initial n'est pas modifié). Cette version est en général considérée comme naïve.

Q11 Écrire une fonction `partitionne(t)` prenant en argument un tableau t et retournant un triplet (v, t_1, t_2) où v est la valeur de $t[0]$, t_1 le tableau des éléments de t strictement inférieurs à v, et t_2 le tableau des éléments de t supérieurs ou égaux à v, $t[0]$ excepté.

Q12 Écrire une fonction `tri_pivot(t)` prenant en argument un tableau t et le triant par la méthode du tri pivot naïf : on partitionne t pour obtenir (v, t_1, t_2), on trie récursivement t_1 et t_2 pour obtenir respectivement u_1 et u_2, et on retourne $u_1 + [v] + u_2$.

Vous testerez votre fonction sur le tableau $[10, 10, 10]$ ainsi que sur les mêmes tableaux que pour le tri par insertion.

