J1MI2013: Algorithmes et Programmes: feuille 10

Dichotomie

Travaux dirigés

Exercice 1. Recherche dans un tableau trié. Soit `t` un tableau de `n` entiers, trié dans l'ordre croissant. On cherche si un entier `x` est présent ou non dans le tableau.

Le principe de la recherche dichotomique est le suivant : on compare `x` avec l'élément médian du tableau `m` (pour le moment, `m=t[n/2]`), puis

- si `m` est égal à `x` alors `x` est présent dans le tableau.
- sinon
  - ou bien `m` est (strictement) plus grand que `x`, et on recherche `x` dans la première moitié du tableau (indices strictement inférieurs à `n/2`),
  - ou bien `m` est (strictement) plus petit que `x` et on recherche `x` dans la seconde moitié du tableau (indices strictement supérieurs à `n/2`).

La recherche de `x` dans le demi-tableau retenu se fait en appliquant le même procédé.

L'algorithme s'arrête lorsque l'élément médian est égal à `x` ou que la partie de tableau dans laquelle s'effectue la recherche est vide.

1. Comment est calculé l'élément médian lorsque la recherche se fait non pas dans le tableau complet, mais dans le sous-tableau entre les indices `debut` et `fin` (inclus) ?
2. Écrire la suite des couples de positions (`debut`, `fin`) délimitant le sous-tableau dans lequel est cherché l'élément lorsque le tableau initial est `[0,1,2,3,4,5,6,7,8,9]` et que
   a. l'élément à chercher est 4.
   b. l'élément à chercher est 1.
   c. l'élément à chercher est -1.
   d. l'élément à chercher est 8.
3. Dans le cas où la valeur cherchée n'est pas présente, comment évolue la taille du tableau dans lequel se fait la recherche lorsque la taille du tableau de départ est $n = 2^k - 1$?
4. Combien de comparaisons entre `x` et des valeurs du tableau sont-elles nécessaires, dans le cas le pire (quel est-il ?), en fonction de `k` ? En fonction de `n` ?
5. Dans le cas général (`n` n'est pas nécessairement de la forme $2^k - 1$), évaluer le nombre de comparaisons (dans le pire des cas) à effectuer lors de la recherche dichotomique, en fonction de la taille `n` du tableau.
6. Écrire la fonction `int rechercheDichotomique(int x, int t[], int n)` qui effectuera la recherche dichotomique de la valeur `x` dans le tableau `t` en appliquant un algorithme **itératif**. Cette fonction retournera un indice `i` tel que `t[i]` vaut `x`, ou `-1` si la valeur `x` n'est pas présente dans le tableau `t`.

7. On souhaite maintenant écrire une version récursive pour le même problème :

   – écrire la fonction **récursive** `int positionRec(int x, int t[], int debut, int fin)` qui retourne un indice `i` compris entre `debut` et `fin` tel que `t[i]` soit égal à `x` si cet indice existe et `-1` sinon ;

   – écrire aussi une fonction `int position(int x, int t[], int n)` qui appellera `positionRec` et renverra la même chose que `rechercheDichotomique`.

Exercice 2. Résolution numérique approchée d'équations.

Soit f une fonction mathématique continue d'un intervalle $[a, b]$ dans $\mathbb{R}$, telle que $f(a)$ et $f(b)$ sont de signes contraires : d'après le théorème des valeurs intermédiaires, il existe au moins un réel x dans l'intervalle $]a, b[$ tel que $f(x) = 0$.

Dans cet exercice, on ne prétend pas trouver **toutes** les solutions de l'équation, qui peut en avoir plusieurs, mais seulement **une** solution (approchée), dont l'existence est garantie par le théorème des valeurs intermédiaires.

Soit $m = \frac{a+b}{2}$ le milieu de l'intervalle : si $f(m) = 0$, alors m est une solution de l'équation. Sinon $f(m)$ a un signe strict, et

– ou bien $f(m)$ et $f(a)$ sont de signes contraires, et l'équation a une solution dans $]a, m[$;

– ou bien $f(m)$ et $f(b)$ sont de signes contraires, et l'équation a une solution dans $]m, b[$.

Pour trouver une solution à ε près de l'équation $f(x) = 0$, il suffit de diviser par deux l'intervalle de recherche jusqu'à ce que sa taille soit inférieure à ε. On suppose déjà écrite en C une fonction `double f(double x)` implémentant la fonction f.

1. Écrire une suite d'instructions pour déterminer une solution approchée à ε près de l'équation $f(x) = 0$ dans l'intervalle $]a, b[$.
2. Évaluer, en fonction de a, b et ε, le nombre maximum d'appels à la fonction `f` que nécessite l'exécution de l'algorithme.

Exercice complémentaire 1. Résolution d'équations sans intervalle initial

Dans l'exercice précédent, on a supposé qu'on connaissait un intervalle fini $]a, b[$ où la fonction f changeait de signe.

Il est possible d'étendre la méthode et de s'affranchir de cette condition, si la fonction f est définie et continue sur $\mathbb{R}$ tout entier et (pour simplifier) croissante sur $\mathbb{R}$, pour peu qu'on ait la certitude que l'équation $f(x) = 0$ admette une solution, *i.e.* $\lim_{x\to-\infty} f(x) < 0$ et $\lim_{x\to+\infty} f(x) > 0$.

La stratégie est la suivante : partant d'un intervalle fini quelconque $[a, b]$ (par exemple $[0, 1]$), si $f(a)$ et $f(b)$ sont de signes contraires on applique la méthode de l'exercice précédent ; si $f(a)$ et $f(b)$ sont du même signe, on peut changer l'intervalle pour un intervalle de longueur double, et répéter ce processus jusqu'à ce que f prenne des valeurs de signes opposés aux extrêmités de cet intervalle.

1. Préciser cette stratégie en écrivant une suite d'instructions permettant de trouver une solution approchée à ϵ près de n'importe quelle équation $f(x) = 0$, pour peu que la fonction f soit continue et strictement croissante sur $\mathbb{R}$, avec $\lim_{x\to-\infty} f(x) < 0$ et $\lim_{x\to+\infty} f(x) > 0$.

2. Donner une estimation du nombre d'utilisations de la fonction `f` nécessaires, en fonction de `ε` et de la valeur de la solution de l'équation.

Exercice complémentaire 2. Adapter l'algorithme de recherche dichotomique dans un tableau trié, pour assurer que l'indice retourné lorsque la valeur cherchée est présente est toujours le *plus petit* indice d'une case contenant la valeur cherchée. Votre modification ne doit pas augmenter significativement la complexité de l'algorithme (en particulier, si le tableau contient un grand nombre de fois la valeur cherchée).