Recherche séquentielle dans un tableau [re03] Exercice

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Version 21 mai 2018

Table des matières

C - Recherche dans un tableau (Solution)

Mots-Clés Recherches ■
Requis Axiomatique impérative sauf Fichiers ■
Fichiers UtilsTB ■
Difficulté • ○ ○ (30 min) ■

Objectif
Cet exercice réalise l'algorithme de la recherche séquentielle dans un tableau. Dans le même ordre d'idées, l'exercice @[Recherche dichotomique dans un tableau] construit un algorithme efficace dans le cas où le tableau est trié.

AJOUT APPARTENANCE JUIN 2017 – A INTEGRE DANS EXO 1 en SIMPLIFIANT la partie II ■

1 Recherche séquentielle / pgrechtab

On considère un tableau d'entiers `t` de `n` éléments. On cherche à construire un algorithme permettant de savoir à quel endroit se trouve une valeur `x`.

Téléchargez le fichier suivant et mettez-le dans votre dossier.

C @[UtilsTB.c]

Copiez/collez ensuite les lignes suivantes :

C **Au début** de votre programme :

```c
#include "UtilsTB.c"
```

Définitions C

```c
enum { TMAX = ... };
typedef int ITableau[TMAX];
```

On suppose que `x| est dans le \lstinline`Tableau@.

Écrivez une fonction `rechSeq1(t,n,x)` qui effectue une recherche séquentielle de `x` (entier) parmi les `n` éléments d'un `Tableau t` et qui renvoie l'indice de sa première occurrence.

Validez votre fonction avec la solution.

Solution C @[pgrechtab.c]

```c
int rechSeq1(const Tableau t, int n, int x)
{
  int ix = 0;
  while (t[ix] != x)
  {
    ix += 1;
  }
  return (ix);
}
```

Exécutez l'algorithme avec les données suivantes :
`n=10`, `t=[1,7,8,9,12,15,18,22,30,31]` et `x=18`.

Solution simple

Indiquons `t`, `ix` et la comparaison `t[ix]<>x` dans le tableau (F signifie `Faux`, V signifie `Vrai`).

t	1	7	8	9	12	15	18	22	30	31
ix	1	2	3	4	5	6	7			
t[ix]<>x	V	V	V	V	V	V	F			

Comment faut-il modifier l'algorithme si l'on n'est pas sûr que la valeur `x` appartienne au tableau ?

Solution simple

Il faut ajouter la possibilité de terminer la boucle si l'on est arrivé au dernier élément du tableau.

Déduisez une fonction `rechSeq2(t,n,x)` qui résout le problème dans le cas général.

Validez votre fonction avec la solution.

Solution C @[pgrechtab.c]

```c
int rechSeq2(const Tableau t, int n, int x)
{
  bool trouve = false;
  int ix = 0;
  while (ix < n && !trouve)
  {
    if (t[ix] == x)
    {
      trouve = true;
    }
    else
    {
      ix += 1;
    }
  }
  return (trouve ? ix : -1);
}
```

On suppose le tableau `t` trié.

Écrivez une fonction `rechSeq3(t,n,x)` qui effectue une recherche séquentielle de `x` parmi les `n` éléments d'un `Tableau` **trié** `t` et qui renvoie l'indice de sa première occurrence.

Solution simple

On peut tirer profit du caractère trié à deux niveaux :

- L'initialisation de l'indice évite le parcours du tableau dans le cas où `x>t[n]`.
- La possibilité de terminer la boucle lorsque l'on tombe sur un élément plus grand que `x` ou si l'on est arrivé au dernier élément du tableau et que celui-ci est inférieur à `x`.

Validez votre fonction avec la solution.

Solution C @[pgrechtab.c]

```c
int rechSeq3(const Tableau* t, int n, int x)
{
  int ix = (t[n-1] < x ? n-1 : -1);
  ix += 1;
  while (ix < n && t[ix] < x)
  {
    ix += 1;
  }
  return (ix < n && t[ix] == x ? ix : -1);
}
```

2 Appartenance d'un élément / pgrechtab2

Soient n un entier positif, t un tableau d'au moins n éléments et e une variable de même type T que les éléments de t. Cet exercice détermine si e est dans t.

2.1 Recherche usuelle

Écrivez une fonction `position1(t,n,e)` qui recherche un entier de valeur `e` parmi les `n` éléments d'un tableau d'entiers `t` et renvoie son indice le plus à gauche (s'il existe), -1 sinon.

Déduisez une fonction `appartenance1(t,n,e)` qui renvoie `Vrai` si un entier de valeur `e` est parmi les `n` éléments d'un tableau d'entiers `t`, `Faux` sinon.

Solution alg

```
Fonction appartenance1(t,n,e)
  Retourner (position1(t,n,e) <> -1)
Finfonc
```

Solution commentée

La fonction se contente d'invoquer la fonction de localisation et de tester si ce n'est pas la valeur -1 (signifiant le cas d'échec).

Calculez la complexité au pire de la fonction d'appartenance.

Solution simple

Pour chaque itération, on effectue deux comparaisons (celle du `Pour` et celle du `Si`) et une incrémentation. Si on effectue n itérations, on aura donc $2n + 1$ comparaisons, n incrémentations et 1 affectation (celle du `Pour`).

2.2 Recherche optimisée avec sentinelle

Dans ce problème, on range en $t[n + 1]$ la valeur e.

Écrivez une fonction `position2(t,n,e)` qui effectue la recherche (avec sentinelle $t[n+1] = e$ (qui existe)) d'un entier de valeur `e` parmi les `n` éléments d'un tableau d'entiers `t` et renvoie son indice le plus à gauche (s'il existe), -1 sinon.

Déduisez une fonction `appartenance2(t,n,e)` de l'appartenance de `e` dans `(t,n)`.

Calculez la complexité de la fonction d'appartenance.

3 Références générales

Comprend [Maunoury-AL1 :c7 :ex6..ex7], [Moliner-ML1 :c2 :ex5, c2 :ex10] ■