Algorithmes de recherche [re]
Algorithmique

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Version 21 mai 2018

Table des matières

Algorithmes de recherche

Mots-Clés Recherche ■
Requis Axiomatique impérative, Récursivité des actions, Complexité des algorithmes ■
Difficulté ● ● ○

Introduction

Ce module étudie les algorithmes de recherche dans un multi-ensemble (il peut y avoir plusieurs éléments ayant la même valeur) muni d'un *ordre total*, c.-à-d. en comparant deux éléments quelconques on peut toujours décider si le premier est strictement plus petit que le deuxième, ou strictement plus grand, ou égal.

En procédant uniquement par comparaison entre éléments, la recherche peut être réalisée de façon **séquentielle**, **dichotomique** ou **par interpolation**. Pour chacune de ces méthodes, nous explicitons un algorithme (avec éventuellement des raffinements et des optimisations) et montrons aussi différentes implantations (version récursive et/ou version itérative).

1 Présentation du problème

1.1 Les méthodes

Les méthodes présentées ici supposent que le (multi-)ensemble est dans une **structure tabulaire** (tableau ou vecteur).

En procédant uniquement par comparaison entre éléments :

- La méthode **séquentielle** parcourt l'un après l'autre les éléments de la séquence en les comparant à l'élément recherché.
- La méthode **dichotomique**, utilisable uniquement lorsque les éléments sont rangés en ordre (croissant ou décroissant), s'appuie sur cette propriété d'ordre pour éliminer, à chaque comparaison, la moitié des éléments de l'ensemble restant à traiter.
- La méthode **par interpolation** tient compte de la valeur de l'élément recherché pour estimer à quel endroit faire une recherche.
- Les méthodes **auto-adaptatives** modifient la position de l'élément recherché afin de tenter d'améliorer les recherches futures.

1.2 Représentation et type des données

Un point fondamental au niveau de l'implantation des algorithmes est la **représentation** et le **type** des données.

Afin d'abstraire la représentation de notre séquence, nous faisons les déclarations suivantes et définissons des opérations sur nos séquences.

Déclarations

```
Constante MAXELEMS <-...
Type Element
  |  ...
FinType

Type Sequence
  |  elems : Element [ MAXELEMS ]
  |  taille : Entier
FinType
```

Opérations

```
Fonction ieme ( DR A : Sequence ; k : Entier ) : Element
Début
  |  Retourner ( A.elems [ k ] )
Fin

Action fixerIeme ( DR A : Sequence ; k : Entier ; valeur : Element )
Début
  |  A.elems [ k ] <- valeur
Fin
```

Fonction de recherche

Cette représentation des données se traduit au niveau de l'en-tête des algorithmes traitant de la recherche de la position d'un `Element x` dans une `Sequence A` par :

```
Fonction rech ( A : Sequence ; x : Element) : Entier
```

Lorsque l'élément cherché n'est pas trouvé, on décide de renvoyer -1 (qui n'est pas un indice valide).

2 Recherche séquentielle

2.1 Principe

La **recherche séquentielle** (*sequential search* en anglais) parcourt, case après case, les `A.taille` éléments d'une `Sequence A` et s'arrête au premier élément rencontré qui est égal à l'`Element x`. Dans ce cas l'information renvoyée est la position (c.-à-d. l'indice) contenant l'élément recherché.

2.2 Version récursive

La récursion se fait sur la taille (= nombre d'éléments) du sous-tableau :

- Si la taille est 0 : l'élément cherché n'y est pas
- Sinon on compare l'élément cherché avec le premier élément :
  - S'il y a égalité : on renvoie cette position
  - Sinon on recommence sur le sous-tableau privé de son premier élément

Fonction rechSeqRec
(Recherche séquentielle récursive)

```
Fonction rechSeqRec ( DR A : Sequence ; x : Element ) : Entier
Début
  |  Retourner rechSeqGRec ( A , 1 , x )
Fin
Fonction rechSeqGRec ( DR A : Sequence ; k : Entier ; x : Element ) : Entier
Début
  |  Si ( k > A.taille ) Alors
  |    |  Retourner ( - 1 )
  |  Sinon
  |    |  Si ieme ( A , k ) = x Alors
  |    |    |  Retourner ( k )
  |    |  Sinon
  |    |    |  Retourner rechSeqGRec ( A , k + 1 , x )
  |    |  FinSi
  |  FinSi
Fin
```

2.3 Version itérative

La vision itérative repose sur un parcours (de la gauche vers la droite) des cases du tableau :

- Au départ on se positionne sur la première case du tableau
- A chaque itération (tour de boucle) on examine le contenu d'une case :
  - S'il est égal à l'élément cherché : on termine en renvoyant la position
  - Sinon on passe à la case suivante
- Si on traverse tout le tableau : l'élément cherché n'était pas présent

Fonction rechSeq
(Recherche séquentielle itérative)

```
Fonction rechSeq ( DR A : Sequence ; x : Element ) : Entier
Variable k : Entier
Variable trouve : Booléen
Début
  |  trouve <- Faux
  |  k <- 1
  |  TantQue ( Non trouve Et k <= A.taille ) Faire
  |    |  Si ( ieme ( A , k ) = x ) Alors
  |    |    |  trouve <- Vrai
  |    |  Sinon
  |    |    |  k <- k + 1
  |    |  FinSi
  |  FinTantQue
  |  Retourner ( trouve ? k : - 1 )
Fin
```

2.4 Version itérative avec sentinelle

Dans l'algorithme `rechSeq`, le test `k <= n` est effectué à chaque itération alors qu'il est utile uniquement lorsque l'élément cherché *n'est pas* dans le tableau. Pour ne pas avoir à faire ce test, il suffit d'être sûr que l'élément est présent dans la partie du tableau où l'on effectue la recherche. On va donc rajouter la valeur de l'élément à chercher après avoir fait le test avec l'élément « sentinelle » puis on effectue la recherche dans le tableau **modifié**.

Ainsi on finira toujours par trouver l'élément recherché, et il suffira de tester s'il est trouvé dans la case modifiée. Cette astuce est dite « recherche avec sentinelle ». Il faut juste penser à restaurer le contenu de la case sentinelle.

Fonction rechSeqS
(Recherche séquentielle itérative avec sentinelle)

```
Fonction rechSeqS ( DR A : Sequence ; x : Element ) : Entier
Variable k : Entier
Variable n : Entier
Variable aux : Element
Début
  |  n <- A.taille
  |  aux <- ieme ( A , n )
  |  Si ( aux = x ) Alors
  |    |  k <- n
  |  Sinon
  |    |  fixerIeme ( A , n , x )
  |    |  k <- 1
  |    |  TantQue ( ieme ( A , k ) <> x ) Faire
  |    |    |  k <- k + 1
  |    |  FinTantQue
  |    |  Si ( k = n ) Alors
  |    |    |  k <- - 1
  |    |  FinSi
  |    |  fixerIeme ( A , n , aux )
  |  FinSi
  |  Retourner ( k )
Fin
```

2.5 Version récursive avec sentinelle

L'utilisation d'une sentinelle pour la version récursive présente le même avantage en gagnant un test à chaque appel récursif.

Fonction rechSeqRecS
(Recherche séquentielle récursive avec sentinelle)

```
Fonction rechSeqRecS ( DR A : Sequence ; x : Element ) : Entier
Variable k : Entier
Variable n : Entier
Variable aux : Element
Début
  |  n <- A.taille
  |  aux <- ieme ( A , n )
  |  Si ( aux = x ) Alors
  |    |  k <- n
  |  Sinon
  |    |  fixerIeme ( A , n , x )
  |    |  k <- rechSeqGRecS ( A , 1 , x )
  |    |  Si ( k = n ) Alors
  |    |    |  k <- - 1
  |    |  FinSi
  |    |  fixerIeme ( A , n , aux )
  |  FinSi
  |  Retourner ( k )
Fin
Fonction rechSeqGRecS ( DR A : Sequence ; k ; : Entier ; x : Element ) : Entier
Début
  |  Si ( ieme ( A , k ) = x ) Alors
  |    |  Retourner ( k )
  |  Sinon
  |    |  Retourner rechSeqGRecS ( A , k + 1 , x )
  |  FinSi
Fin
```

3 Recherche séquentielle dans une séquence triée

3.1 Principe

On suppose que les éléments du tableau sont triés en ordre croissant.

L'ordre permet de déterminer exactement l'intervalle auquel doit appartenir l'élément cherché ce qui accélère la recherche. Tout d'abord un simple test avec le dernier élément du tableau permet de savoir si l'élément cherché est dans l'intervalle ; et dans ce cas, le tableau contient un élément `z` supérieur ou égal à l'élément cherché. Cet élément `z` sert de sentinelle et l'on peut donc arrêter la recherche dès que l'on rencontre un tel `z` (recherche avec échec si `z` est strictement supérieur à l'élément cherché).

3.2 Version itérative

Fonction rechLin
(Recherche linéaire itérative : traduit le principe)

```
Fonction rechLin ( DR A : Sequence ; x : Element ) : Entier
Variable k : Entier
Début
  |  Si ( ieme ( A , A.taille ) < x ) Alors
  |    |  Retourner ( - 1 )
  |  Sinon
  |    |  k <- 1
  |    |  TantQue ieme ( A , k ) < x Faire
  |    |    |  k <- k + 1
  |    |  FinTantQue
  |    |  Si ( ieme ( A , k ) = x ) Alors
  |    |    |  Retourner ( k )
  |    |  Sinon
  |    |    |  Retourner ( - 1 )
  |    |  FinSi
  |  FinSi
Fin
```

3.3 Version récursive

Fonction rechLinRec
(Recherche linéaire récursive)

```
Fonction rechLinRec ( DR A : Sequence ; x : Element ) : Entier
Début
  |  Si ( ieme ( A , A.taille ) < x ) Alors
  |    |  Retourner ( - 1 )
  |  Sinon
  |    |  Retourner ( rechLinGRec ( A , 1 , n , x ) )
  |  FinSi
Fin
Fonction rechLinGRec ( DR A : Sequence ; k ; : Entier ; x : Element ) : Entier
Début
  |  Si ( ieme ( A , k ) < x ) Alors
  |    |  Retourner rechLinGRec ( A , k + 1 , x )
  |  Sinon
  |    |  Si ( ieme ( A , k ) = x ) Alors
  |    |    |  Retourner ( k )
  |    |  Sinon
  |    |    |  Retourner ( - 1 )
  |    |  FinSi
  |  FinSi
Fin
```

4 Recherche dichotomique

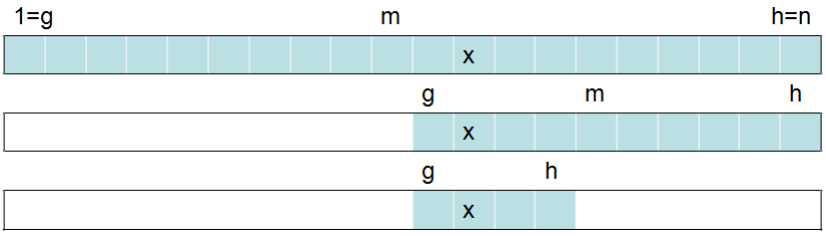
4.1 Principe de la recherche dichotomique

La recherche séquentielle dans une séquence triée tient très peu compte de la croissance des éléments de la séquence. En revanche la recherche par dichotomie utilise cette information sur l'ordre des éléments de manière beaucoup plus efficace.

Recherche dichotomique

La **recherche par dichotomie** (*binary search* en anglais) compare l'élément cherché `x` avec l'élément en position `m` situé au milieu du sous-tableau :

- si `A[m] = x` : on a trouvé l'élément en position `m`
- si `A[m] > x` : il est impossible que `x` se trouve après la position `m` dans le tableau (puisque les éléments sont en ordre croissant). Il reste à traiter uniquement la moitié inférieure du tableau
- de même si `A[m] < x` : il reste à traiter uniquement la moitié supérieure du tableau
- on continue ainsi la recherche, et après chaque comparaison, la taille du sous-tableau restant à traiter diminue de moitié. Si la recherche aboutit sur un tableau de taille nulle, `x` n'est pas présent et la recherche s'arrête.



Remarque

Si le tableau contient plusieurs fois l'élément cherché, la fonction renverra l'un des indices où il apparaît, mais on ne précise pas laquelle.

4.2 Version récursive

Fonction rechDichoRec

(Recherche dichotomique récursive : suit directement la description)

```
Fonction rechDichoRec ( DR A : Sequence ; x : Element ) : Entier
Début
  |  Retourner rechDichoAuxRec ( A , 1 , A.taille , x )
Fin
Fonction rechDichoAuxRec ( DR A : Sequence ; g , h : Entier ; x : Element ) : Entier
Début
  |  Si ( h < g ) Alors
  |    |  Retourner ( - 1 )
  |  Sinon
  |    |  m <- DivEnt ( g + h , 2 )
  |    |  Si ( ieme ( A , m ) = x ) Alors
  |    |    |  Retourner ( m )
  |    |  Sinon
  |    |    |  Si ( ieme ( A , m ) < x ) Alors
  |    |    |    |  Retourner rechDichoAuxRec ( A , m + 1 , h , x )
  |    |    |  Sinon
  |    |    |    |  Retourner rechDichoAuxRec ( A , g , m - 1 , x )
  |    |    |  FinSi
  |    |  FinSi
  |  FinSi
Fin
```

4.3 Version itérative

La version itérative transcrit la version récursive en modifiant l'une des deux extrémités du sous-tableau.

Fonction rechDicho
(Recherche dichotomique itérative)

```
Fonction rechDicho ( A : Sequence ; x : Element ) : Entier
Variable g , h , m : Entier
Variable trouve : Booléen
Début
  |  trouve <- Faux
  |  g <- 1
  |  h <- A.taille
  |  TantQue ( g <= h Et Non trouve ) Faire
  |    |  m <- DivEnt ( g + h , 2 )
  |    |  Si ( ieme ( A , k ) = x ) Alors
  |    |    |  trouve <- Vrai
  |    |  Sinon
  |    |    |  Si ( ieme ( A , k ) < x )
  |    |    |    |  g <- m + 1
  |    |    |  Sinon
  |    |    |    |  h <- m - 1
  |    |    |  FinSi
  |    |  FinSi
  |  FinTantQue
  |  Si trouve Alors
  |    |  Retourner ( m )
  |  Sinon
  |    |  Retourner ( - 1 )
  |  FinSi
Fin
```

5 Recherche par interpolation

5.1 Principe

La **recherche par interpolation** tient compte de la *valeur* de l'élément recherché pour estimer à quel endroit faire une recherche. C'est le principe de la recherche dans un dictionnaire : pour rechercher le mot « ananas », on ouvrira le dictionnaire vers le début, plutôt qu'au milieu.

Plus précisément, supposons que les éléments de la séquence sont des nombres, rangés en ordre croissant de l'indice `ideb` à l'indice `ifin`, et `A[k]` le `k`-ème élément de la séquence. Sous l'hypothèse que les nombres sont en progression régulière, on admettra qu'il paraît raisonnable d'aller chercher le nombre `x` autour de la position `p` telle que :

$$\frac{p - ideb}{ifin - ideb + 1} = \frac{x - A[ideb]}{A[ifin] - A[ideb] + 1}$$

La valeur de l'indice `p` est obtenue en prenant la partie entière de la solution pour p dans l'équation ci-dessus. Cette valeur est valide, c.-à-d. comprise entre les bornes `ideb` à l'indice `ifin`, uniquement lorsque `x` est compris entre les éléments `A[ideb]` et `A[ifin]` de la séquence.

Lorsque l'indice `p` est valide, on compare `x` et `A[p]` :

- si `x = A[p]` : on a terminé
- si `x < A[p]` : on recommence la recherche sur `A[ideb..p-1]`
- si `x > A[p]` : on recommence la recherche sur `A[p+1..ifin]`

Lorsque l'indice `p` n'est pas valide, l'élément `x` n'est pas dans la séquence.

5.2 Version itérative

Fonction rechInterpol

(Recherche par interpolation itérative : transcrit le principe)

```
Fonction rechInterpol ( DR A : Sequence ; x : Element ) : Entier
Variable g , h , p : Entier
Variable trouve : Booléen
Variable numer , denom : Entier
Début
  |  trouve <- Faux
  |  g <- 1
  |  h <- A.taille
  |  TantQue ( g <= h Et Non trouve ) Faire
  |    |  Si ( x < ieme ( A , g ) Ou ieme ( A , h ) < x ) Alors
  |    |    |  Sortir
  |    |  FinSi
  |    |  numer <- x - ieme ( A , g )
  |    |  denom <- ieme ( A , h ) - ieme ( A , g ) + 1
  |    |  p <- g + DivEnt ( ( h - g + 1 ) * numer , denom )
  |    |  Si ( ieme ( A , p ) = x ) Alors
  |    |    |  trouve <- Vrai
  |    |  Sinon
  |    |    |  Si ( ieme ( A , p ) < x )
  |    |    |    |  g <- p + 1
  |    |    |  Sinon
  |    |    |    |  h <- p - 1
  |    |    |  FinSi
  |    |  FinSi
  |  FinTantQue
  |  Si trouve Alors
  |    |  Retourner ( p )
  |  Sinon
  |    |  Retourner ( - 1 )
  |  FinSi
Fin
```

6 PREVU : Recherches auto-adaptatives

7 Complexités

7.1 Recherche séquentielle

On note $C(n)$ le nombre de tests maximum pour rechercher un élément dans un tableau de taille n. Dans tous les cas, le coût maximum est en $O(n)$.

7.2 Recherche linéaire

7.3 Recherche dichotomique

Le principe de la méthode est, à chaque étape, de tirer profit de la comparaison entre l'élément cherché et l'élément du milieu du tableau pour couper le tableau en deux, d'où son nom « dicho(deux)-tomie(couper) ».

Lorsque n est une puissance de 2, le nombre de fois que l'on peut diviser n par 2 avant d'arriver sur la valeur 1 est exactement $\lg n$:

$$n = 2^p \Rightarrow p = \lg n$$

Et plus généralement pour un entier n quelconque, le nombre de divisions entières successives de n par 2 avant d'atteindre 1 est au plus $\lfloor \lg n \rfloor$.

Application
Pour des tableaux de petits taille la différence d'efficacité par rapport à la méthode séquentielle n'est pas essentielle mais lorsque n devient grand, le gain devient considérable. Exemple : pour un tableau d'un million d'éléments, la recherche séquentielle peu nécessiter jusqu'à deux millions de tests (sans sentinelle). Alors que la recherche dichotomique n'en fera au plus que quelques dizaines puisque $10^6 \approx 2^{20}$ et donc $\lg 10^6 \approx 20$.

7.4 Recherche par interpolation

On peut montrer que le nombre moyen de comparaisons pour une recherche par interpolation dans une séquence de n éléments est très proche de $\lg \lg n$.