Algorithmes de recherche

Ch. Lasou, N.E. Oussous, E. Wegrzynowski

Licence ST-A, USTL - API1

5 février 2007

1. Introduction
2. Recherche séquentielle
   - dans un tableau non trié
3. Recherche séquentielle
4. Recherche dichotomique

Le problème

Problème de la recherche

Données $t[a..b]$ un tableau d'éléments de type E, $x \in E$

But déterminer si $x \in t$

Le problème

Problème de la recherche

Données $t[a..b]$ un tableau d'éléments de type E, $x \in E$

But déterminer si $x \in t$

1. le résultat est de nature booléenne

Le problème

Problème de la recherche

Données $t[a..b]$ un tableau d'éléments de type E, $x \in E$

But déterminer si $x \in t$

1. le résultat est de nature booléenne
   - `vrai` si $x \in t$: recherche avec succès

Le problème

Problème de la recherche

Données $t[a..b]$ un tableau d'éléments de type E, $x \in E$

But déterminer si $x \in t$

1. le résultat est de nature booléenne
   - `vrai` si $x \in t$: recherche avec succès
   - `faux` si $x \notin t$: recherche avec échec

Le problème

Problème de la recherche

Données $t[a..b]$ un tableau d'éléments de type E, $x \in E$

But déterminer si $x \in t$

1. le résultat est de nature booléenne
   - `vrai` si $x \in t$: recherche avec succès
   - `faux` si $x \notin t$: recherche avec échec
2. il peut aussi être accompagné d'un indice $i \in [\![a, b]\!]$ tel que $t[i] = x$ s'il en existe (et dans ce cas lequel ? le plus petit ? le plus grand ?...)

Objectifs

- déterminer des schémas d'algorithmes de recherche

Objectifs

- déterminer des schémas d'algorithmes de recherche
- en distinguant deux cas

Objectifs

- déterminer des schémas d'algorithmes de recherche
- en distinguant deux cas
  1. les tableaux quelconques

Objectifs

- déterminer des schémas d'algorithmes de recherche
- en distinguant deux cas
  1. les tableaux quelconques
  2. les tableaux triés

Objectifs

- déterminer des schémas d'algorithmes de recherche
- en distinguant deux cas
  1. les tableaux quelconques
  2. les tableaux triés

Dans tout le chapitre on suppose que les tableaux sont indexés par des entiers. Il est possible d'adapter les algos au cas de tableaux indexés par d'autres types ordinaux.

Principe de la recherche séquentielle

- recherche de x en parcourant les éléments de t un à un

Principe de la recherche séquentielle

- recherche de x en parcourant les éléments de t un à un
- par ordre croissant (ou bien décroissant) des indices

Principe de la recherche séquentielle

- recherche de x en parcourant les éléments de t un à un
- par ordre croissant (ou bien décroissant) des indices
- jusqu'au premier indice k tel que $t[k] = x$: RECHERCHE AVEC SUCCÈS

Principe de la recherche séquentielle

- recherche de x en parcourant les éléments de t un à un
- par ordre croissant (ou bien décroissant) des indices
- jusqu'au premier indice k tel que $t[k] = x$: RECHERCHE AVEC SUCCÈS
- ou bien jusqu'à avoir parcouru l'intervalle $[\![a, b]\!]$ en entier sans trouver x : RECHERCHE AVEC ÉCHEC

Vers l'algorithme

Hypothèse : Soit $k \in [\![a, b]\!]$ et supposons que $x \notin t[a..k-1]$

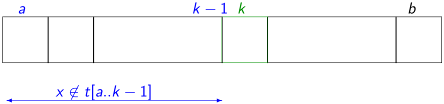


FIG.: recherche séquentielle

Vers l'algorithme

Hypothèse : Soit $k \in [\![a, b]\!]$ et supposons que $x \notin t[a..k-1]$

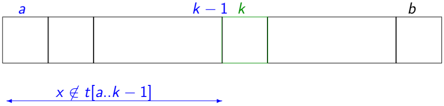


Fig.: recherche séquentielle

- Si $t[k] = x$, alors recherche terminée avec SUCCÈS

Vers l'algorithme

Hypothèse : Soit $k \in [\![a, b]\!]$ et supposons que $x \notin t[a..k-1]$

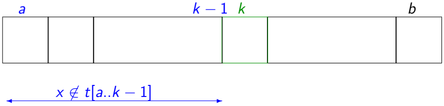


Fig.: recherche séquentielle

- Si $t[k] = x$, alors recherche terminée avec SUCCÈS
- Si $t[k] \neq x$, alors passer à l'élément suivant

Vers l'algorithme

Hypothèse : Soit $k \in [\![a, b]\!]$ et supposons que $x \notin t[a..k-1]$

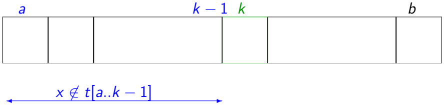


FIG.: recherche séquentielle

- Si $t[k] = x$, alors recherche terminée avec SUCCÈS
- Si $t[k] \neq x$, alors passer à l'élément suivant
  - s'il y en a un !

Vers l'algorithme

Hypothèse : Soit $k \in [\![a, b]\!]$ et supposons que $x \notin t[a..k-1]$

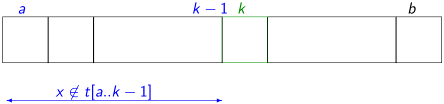


FIG.: recherche séquentielle

- Si $t[k] = x$, alors recherche terminée avec SUCCÈS
- Si $t[k] \neq x$, alors passer à l'élément suivant
  - s'il y en a un !
  - sinon recherche terminée avec ÉCHEC

Algorithme version 1

```
k := a
{x ∉ t[a..k − 1]}
tant que k ≤ b et t[k] ≠ x faire
   {x ∉ t[a..k]}
   inc(k)
   {x ∉ t[a..k − 1]}
fin tant que
{(k > b et x ∉ t),
 ou bien (k ≤ b et t[k] = x)}
si k > b alors
   recherche avec ÉCHEC
sinon
   recherche avec SUCCÈS,
   et k est le plus petit indice tq t[k] = x
```

Remarque

Dans le cas d'une recherche qui échoue, la condition du **tant que** de l'algorithme qui précède est correctement exprimée ou non selon la nature de l'opérateur **et** utilisé dans le langage de programmation.

1. correct si l'opérateur est séquentiel, i.e. évaluation partielle des termes de la conjonction

a	b	a et alors b
`vrai`	`vrai`	`vrai`
`vrai`	`faux`	`faux`
`faux`	`*`	`faux`

équivalent à

```
si a alors
  b
sinon
  faux
```

1. correct si l'opérateur est <u>séquentiel</u>, i.e. évaluation partielle des termes de la conjonction

a	b	a et alors b
`vrai`	`vrai`	`vrai`
`vrai`	`faux`	`faux`
`faux`	`*`	`faux`

équivalent à

```
si a alors
  b
sinon
  faux
```

2. incorrect sinon, i.e. évaluation complète des termes de la conjonction.

1. correct si l'opérateur est séquentiel, i.e. évaluation partielle des termes de la conjonction

a	b	a et alors b
vrai	vrai	vrai
vrai	faux	faux
faux	*	faux

équivalent à

```
si a alors
  b
sinon
  faux
```

2. incorrect sinon, i.e. évaluation complète des termes de la conjonction.

Avec Free Pascal, l'opérateur **and** est séquentiel.

Algorithme version 2

Algorithme de recherche séquentiel convenable avec opérateur **et** non séquentiel.

```
k := a
{x ∉ t[a..k − 1]}
tant que k < b et t[k] ≠ x faire
   {x ∉ t[a..k]}
   inc(k)
   {x ∉ t[a..k − 1]}
fin tant que
{(k = b et x ∉ t[a..b − 1]),
 ou bien (k < b et t[k] = x)}
si t[k] = x alors
   recherche avec SUCCÈS,
   et k est le plus petit indice tq t[k] = x
sinon
   recherche avec ÉCHEC
```

Coût de la recherche séquentielle

Nombre de comparaisons ($t[k] = x$) effectuées dans la recherche séquentielle dans un tableau de n éléments

Coût de la recherche séquentielle

Nombre de comparaisons ($t[k] = x$) effectuées dans la recherche séquentielle dans un tableau de n éléments

1. pour une recherche qui échoue : $c(n) = n$

Coût de la recherche séquentielle

Nombre de comparaisons ($t[k] = x$) effectuées dans la recherche séquentielle dans un tableau de n éléments

1. pour une recherche qui échoue : $c(n) = n$
2. pour une recherche qui réussit, cela dépend

Coût de la recherche séquentielle

Nombre de comparaisons ($t[k] = x$) effectuées dans la recherche séquentielle dans un tableau de n éléments

1. pour une recherche qui échoue : $c(n) = n$
2. pour une recherche qui réussit, cela dépend
   - $c(n) = 2$ si $t[1] = x$: meilleur des cas

Coût de la recherche séquentielle

Nombre de comparaisons ($t[k] = x$) effectuées dans la recherche séquentielle dans un tableau de n éléments

1. pour une recherche qui échoue : $c(n) = n$
2. pour une recherche qui réussit, cela dépend
   - $c(n) = 2$ si $t[1] = x$: <u>meilleur des cas</u>
   - $c(n) = n$ si $t[n] = x$: <u>pire des cas</u>

Coût de la recherche séquentielle

Nombre de comparaisons ($t[k] = x$) effectuées dans la recherche séquentielle dans un tableau de n éléments

1. pour une recherche qui échoue : $c(n) = n$
2. pour une recherche qui réussit, cela dépend
   - $c(n) = 2$ si $t[1] = x$: meilleur des cas
   - $c(n) = n$ si $t[n] = x$: pire des cas
   - $1 \leq c(n) \leq n$ dans tous les cas

Tableaux triés

Définition

Supposons le type E totalement ordonné par la relation notée $\leq$.
Un tableau $t[a..b]$ d'éléments de type E est <u>trié</u> pour l'ordre $\leq$ si

$$\forall i \in [\![a, b-1]\!] \quad t[i] \leq t[i+1]$$

Tableaux triés

Définition

Supposons le type E totalement ordonné par la relation notée $\leq$.
Un tableau $t[a..b]$ d'éléments de type E est trié pour l'ordre $\leq$ si

$$\forall i \in [\![a, b-1]\!] \quad t[i] \leq t[i+1]$$

Exemple

Exemple de tableau d'entiers trié pour l'ordre usuel

1	2	3	4	5	6
0	5	5	7	12	21

Principe de l'algorithme

- recherche de x en parcourant les éléments de t un à un

Principe de l'algorithme

- recherche de x en parcourant les éléments de t un à un
- par ordre croissant des indices

Principe de l'algorithme

- recherche de x en parcourant les éléments de t un à un
- par ordre croissant des indices
- jusqu'à atteindre un indice k tel que $x \leq t[k]$:

Principe de l'algorithme

- recherche de x en parcourant les éléments de t un à un
- par ordre croissant des indices
- jusqu'à atteindre un indice k tel que $x \leq t[k]$:
  - si $x = t[k]$: RECHERCHE AVEC SUCCÈS

Principe de l'algorithme

- recherche de x en parcourant les éléments de t un à un
- par ordre croissant des indices
- jusqu'à atteindre un indice k tel que $x \leq t[k]$:
  - si $x = t[k]$: RECHERCHE AVEC SUCCÈS
  - sinon, $x < t[k]$:RECHERCHE AVEC ÉCHEC

Principe de l'algorithme

- recherche de x en parcourant les éléments de t un à un
- par ordre croissant des indices
- jusqu'à atteindre un indice k tel que $x \leq t[k]$:
  - si $x = t[k]$: RECHERCHE AVEC SUCCÈS
  - sinon, $x < t[k]$:RECHERCHE AVEC ÉCHEC
- ou bien jusqu'à avoir parcouru l'intervalle $[\![a, b]\!]$ en entier sans trouver x : RECHERCHE AVEC ÉCHEC

Recherche séquentielle

dans un tableau trié

```
k := a
{x ∉ t[a..k − 1]}
tant que k < b et t[k] < x faire
   {x ∉ t[a..k]}
   inc(k)
   {x ∉ t[a..k − 1]}
fin tant que
{(k = b et x ∉ t[a..b − 1]),
 ou bien (k < b et t[k] ≥ x)}
si t[k] = x alors
   recherche avec SUCCÈS,
   et k est le plus petit indice tq t[k] = x
sinon
   recherche avec ÉCHEC
```

Coût de la recherche séquentielle

Nombre de comparaisons ($t[k] = x$) effectuées dans la recherche séquentielle dans un tableau trié de n éléments

Coût de la recherche séquentielle

Nombre de comparaisons ($t[k] = x$) effectuées dans la recherche séquentielle dans un tableau trié de n éléments

1. pour une recherche qui échoue, cela dépend

Coût de la recherche séquentielle

Nombre de comparaisons ($t[k] = x$) effectuées dans la recherche séquentielle dans un tableau trié de n éléments

1. pour une recherche qui échoue, cela dépend
   - $c(n) = 2$ si $t[1] > x$: <u>meilleur des cas</u>

Coût de la recherche séquentielle

Nombre de comparaisons ($t[k] = x$) effectuées dans la recherche séquentielle dans un tableau trié de n éléments

1. pour une recherche qui échoue, cela dépend
   - $c(n) = 2$ si $t[1] > x$: <u>meilleur des cas</u>
   - $c(n) = n$ si $t[n] < x$: <u>pire des cas</u>

Coût de la recherche séquentielle

Nombre de comparaisons ($t[k] = x$) effectuées dans la recherche séquentielle dans un tableau trié de n éléments

1. pour une recherche qui échoue, cela dépend
   - $c(n) = 2$ si $t[1] > x$: meilleur des cas
   - $c(n) = n$ si $t[n] < x$: pire des cas
   - $1 \leq c(n) \leq n$ dans tous les cas

Coût de la recherche séquentielle

Nombre de comparaisons ($t[k] = x$) effectuées dans la recherche séquentielle dans un tableau trié de n éléments

1. pour une recherche qui échoue, cela dépend
   - $c(n) = 2$ si $t[1] > x$: meilleur des cas
   - $c(n) = n$ si $t[n] < x$: pire des cas
   - $1 \leq c(n) \leq n$ dans tous les cas
2. pour une recherche qui réussit, cela dépend

Coût de la recherche séquentielle

Nombre de comparaisons ($t[k] = x$) effectuées dans la recherche séquentielle dans un tableau trié de n éléments

1. pour une recherche qui échoue, cela dépend
   - $c(n) = 2$ si $t[1] > x$: meilleur des cas
   - $c(n) = n$ si $t[n] < x$: pire des cas
   - $1 \leq c(n) \leq n$ dans tous les cas
2. pour une recherche qui réussit, cela dépend
   - $c(n) = 2$ si $t[1] = x$: meilleur des cas

Coût de la recherche séquentielle

Nombre de comparaisons ($t[k] = x$) effectuées dans la recherche séquentielle dans un tableau trié de n éléments

1. pour une recherche qui échoue, cela dépend
   - $c(n) = 2$ si $t[1] > x$: <u>meilleur des cas</u>
   - $c(n) = n$ si $t[n] < x$: <u>pire des cas</u>
   - $1 \leq c(n) \leq n$ dans tous les cas
2. pour une recherche qui réussit, cela dépend
   - $c(n) = 2$ si $t[1] = x$: <u>meilleur des cas</u>
   - $c(n) = n$ si $t[n] = x$: <u>pire des cas</u>

Coût de la recherche séquentielle

Nombre de comparaisons ($t[k] = x$) effectuées dans la recherche séquentielle dans un tableau trié de n éléments

1. pour une recherche qui échoue, cela dépend
   - $c(n) = 2$ si $t[1] > x$: meilleur des cas
   - $c(n) = n$ si $t[n] < x$: pire des cas
   - $1 \leq c(n) \leq n$ dans tous les cas
2. pour une recherche qui réussit, cela dépend
   - $c(n) = 2$ si $t[1] = x$: meilleur des cas
   - $c(n) = n$ si $t[n] = x$: pire des cas
   - $1 \leq c(n) \leq n$ dans tous les cas

Principe de l'algorithme

- Les idées

Principe de l'algorithme

- Les idées
  - tenir compte de l'ordre des éléments du tableau

Principe de l'algorithme

- Les idées
  - tenir compte de l'ordre des éléments du tableau
  - et de l'accès direct à ces éléments

Principe de l'algorithme

- Les idées
  - tenir compte de l'ordre des éléments du tableau
  - et de l'accès direct à ces éléments
  - pour diviser par deux à chaque étape la taille du tableau à visiter.

Principe de l'algorithme

- Les idées
  - tenir compte de l'ordre des éléments du tableau
  - et de l'accès direct à ces éléments
  - pour diviser par deux à chaque étape la taille du tableau à visiter.
- Le principe de l'algorithme

Principe de l'algorithme

- Les idées
  - tenir compte de l'ordre des éléments du tableau
  - et de l'accès direct à ces éléments
  - pour diviser par deux à chaque étape la taille du tableau à visiter.
- Le principe de l'algorithme
  - calcul de l'indice du milieu $m = \frac{a+b}{2}$

Principe de l'algorithme

- Les idées
  - tenir compte de l'ordre des éléments du tableau
  - et de l'accès direct à ces éléments
  - pour diviser par deux à chaque étape la taille du tableau à visiter.
- Le principe de l'algorithme
  - calcul de l'indice du milieu $m = \frac{a+b}{2}$
  - si $t[m] = x$ alors : RECHERCHE AVEC SUCCÈS

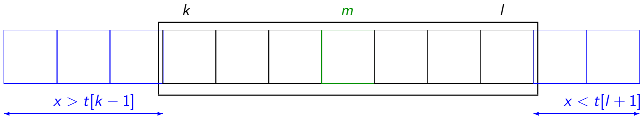
Principe de l'algorithme

- Les idées
  - tenir compte de l'ordre des éléments du tableau
  - et de l'accès direct à ces éléments
  - pour diviser par deux à chaque étape la taille du tableau à visiter.
- Le principe de l'algorithme
  - calcul de l'indice du milieu $m = \frac{a+b}{2}$
  - si $t[m] = x$ alors : RECHERCHE AVEC SUCCÈS
  - sinon si $t[m] < x$ alors recherche dans $t[m+1..b]$

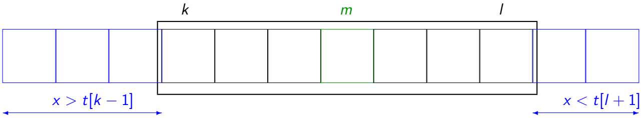
Principe de l'algorithme

- Les idées
  - tenir compte de l'ordre des éléments du tableau
  - et de l'accès direct à ces éléments
  - pour diviser par deux à chaque étape la taille du tableau à visiter.
- Le principe de l'algorithme
  - calcul de l'indice du milieu $m = \frac{a+b}{2}$
  - si $t[m] = x$ alors : RECHERCHE AVEC SUCCÈS
  - sinon si $t[m] < x$ alors recherche dans $t[m+1..b]$
  - sinon recherche dans $t[a..m-1]$

L'algorithme en action (cas 1)



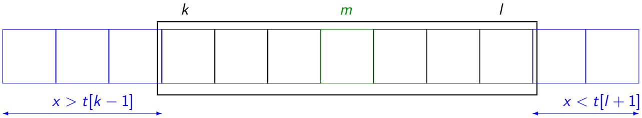
L'algorithme en action (cas 1)



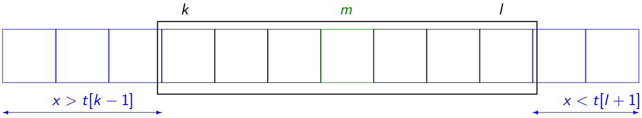
si $t[m] = x$, alors

$$\boxed{\text{Recherche avec succès}}$$

L'algorithme en action (cas 2)

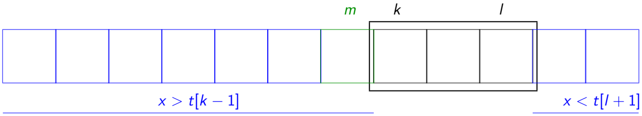


L'algorithme en action (cas 2)

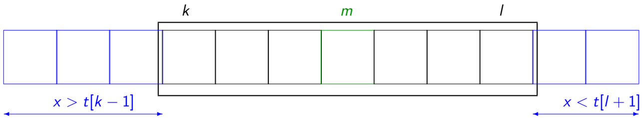


si $t[m] < x$, alors

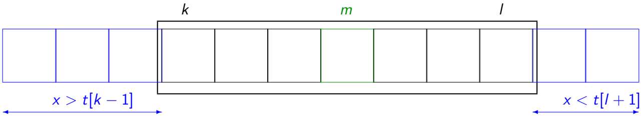
- $k := m + 1$



L'algorithme en action (cas 3)

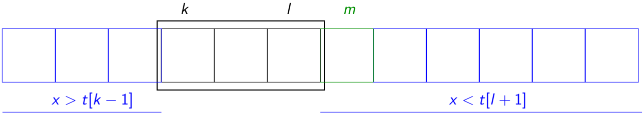


L'algorithme en action (cas 3)



si $t[m] > x$, alors

- $l := m - 1$



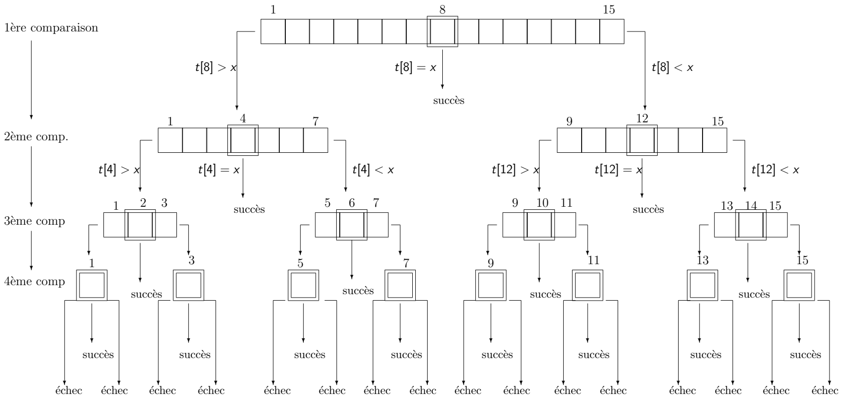
Algorithme

Recherche dichotomique dans un tableau trié

```
k := a
l := b
trouve := faux
{x ∉ t[a..k − 1], x ∉ t[l + 1..b]}
tant que k ≤ l et non trouve faire
  m := (k+l)/2
  si t[m] = x alors
     trouve := vrai
  sinon si t[m] < x alors
     k := m + 1
  sinon
     l := m − 1
  fin si
  {invariant : x ∉ t[a..k − 1], x ∉ t[l + 1..b]}
fin tant que
{(trouve = vrai ou k > l) et x ∉ t[a..k − 1],
 x ∉ t[l + 1..b]}
si trouve alors
  recherche avec SUCCÈS,
sinon
  recherche avec ÉCHEC
```

Coût de la recherche dichotomique

dans un tableau de taille 15



Coût de la recherche dichotomique

Nombre de comparaisons ($t[k] = x$) effectuées dans la recherche dichotomique dans un tableau trié de n éléments

1. Pour une recherche qui échoue, $c(n) = 1 + \log_2(n)$
2. Pour une recherche qui réussit, cela dépend
   - $c(n) = 1$ si l'élément recherché est au milieu du tableau : meilleur des cas
   - $c(n) = 1 + \log_2(n)$ dans le pire des cas
   - $1 \leq c(n) \leq 1 + \log_2(n)$