Algorithmes Avancés

Algorithmes de recherche

Recherche Séquentielle

Utilisation

La **recherche séquentielle** est utilisée pour rechercher une valeur dans un tableau non trié.

Principe

Parcourir le tableau, dans l'ordre, de la première valeur jusqu'à trouver la valeur recherchée ou parcourir tout le tableau.
Si la valeur est retrouvée on retourne son indice dans le tableau ou on peut aussi retourner un booléen pour vérifier l'existence de l'élément.

Simulation

Chercher dans quelle boîte se cache le nombre 338.
338 est trouvée à la position 3.

Travail demandé

Ecrire l'algorithme d'une fonction qui recherche l'existence d'une valeur `v` dans un tableau `t` de `n` éléments.

Solution - Nouveau régime

Algorithme

```
Fonction existe(v: entier; t: tab; n: entier):booléen
Début
  i ← 0
  trouve ← Faux
  TantQue (non trouve) et (i < n) Faire
    trouve ← t[i] = v
    i ← i + 1
  Fin TantQue
  Retourner trouve
Fin
```

T.D.O.L

Objet	Type/Nature
i	entier
trouve	booléen

Solution - Ancien régime

Algorithme

```
DEF FN existe(v: entier; t: tab; n: entier):booléen
Début
  i ← 1
  trouve ← Faux
  TantQue (non trouve) et (i ≤ n) Faire
    trouve ← t[i] = v
    i ← i + 1
  Fin TantQue
  existe ← trouve
Fin
```

T.D.O.L

Objet	Type/Nature
i	entier
trouve	booléen

Recherche Dichotomique

Utilisation

La **Rechercher dichotomique** est utilisée pour retrouver une valeur dans un tableau trié.

Principe

On suppose que la variable **d** indique l'indice du premier élément du tableau, que `f` indique l'indice du dernier élément du tableau et que le tableau **est trié en ordre croissant**.

1. Calculer l'indice `m` de l'élément qui se trouve au centre de l'intervalle `[d, f]` : m ⟵ (d + f) div 2
2. Si `t[m] = v` ⇒ Valeur trouvée à l'indice m
3. Si `t[m] < v` ⇒ Rechercher dans la partie droite du tableau, d ⟵ m + 1
4. Si `t[m] > v` ⇒ Rechercher dans la partie gauche du tableau, f ⟵ m - 1

Simulation

Chercher dans quelle boîte se cache le nombre 606.
606 est trouvée à la position 6.

Travail demandé

Ecrire l'algorithme d'une fonction qui recherche l'existence d'une valeur `v` dans un tableau `t` de `n` éléments.

Solution - Nouveau régime

Algorithme

```
Fonction existe(v: entier; t: tab; n: entier):booléen
Début
  d ← 0 ; f ← n-1
  TantQue f ≥ d Faire
    m ← (d + f) div 2
    Si t[m] = v Alors Retourner Vrai
    Sinon Si t[m] < v Alors d ← m + 1
    Sinon f ← m - 1 Fin Si
  Fin TantQue
  Retourner Faux
Fin
```

T.D.O.L

Objet	Type/Nature
d, f, m	entier

Solution - Ancien régime

Algorithme

```
DEF FN existe(v: entier; t: tab; n: entier):booléen
Début
  d ← 0 ; f ← n-1 ; trouve ← Faux
  TantQue (non trouve) and (f ≥ d) Faire
    m ← (d + f) div 2
    Si t[m] = v Alors trouve ← Vrai
    Sinon Si t[m] < v Alors d ← m + 1
    Sinon f ← m - 1 Fin Si
  Fin TantQue
  existe ← trouve
Fin
```

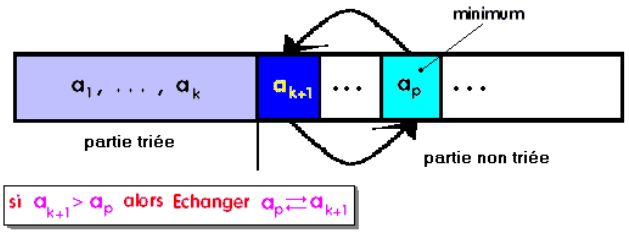
T.D.O.L

Objet	Type/Nature
d, f, m	entier
trouve	booléen

Algorithmes de tri

Tri par sélection

Utilisation



Ordonner les éléments d'un tableau en ordre croissant ou décroissant.

Principe

1. Pour chaque élément d'indice `i` allant de 0 à n-2
2. Pour chaque élément d'indice `j` allant de i+1 à n-1
3. Si t[j] < t[i] ⇒ permuter(t[i], t[j])

Travail demandé

Ecrire l'algorithme de la procédure `TriSelection(n, t)` qui ordonne les éléments de `t` en ordre décroissant.

Solution - Nouveau régime

Algorithme

```
Procédure TriSelection(n: entier; @t: tab)
Début
  Pour i de 0 à n-2 faire
    Pour j de i+1 à n-1 faire
      Si t[j] < t[i] Alors
        aux ← t[j]
        t[j] ← t[i]
        t[i] ← aux
      Fin Si
    Fin Pour
  Fin Pour
Fin
```

T.D.O.L

Objet	Type/Nature
i, j, aux	entier

Solution - Ancien régime

Algorithme

```
Procédure TriSelection(n: entier; var t: tab)
Début
  Pour i de 1 à n-1 faire
    Pour j de i+1 à n faire
      Si t[j] < t[i] Alors
        aux ← t[j]
        t[j] ← t[i]
        t[i] ← aux
      Fin Si
    Fin Pour
  Fin Pour
Fin
```

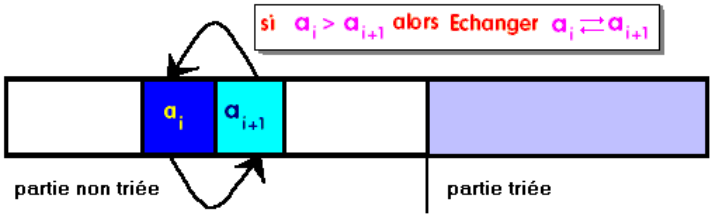
T.D.O.L

Objet	Type/Nature
i, j, aux	entier

Tri à bulles

Utilisation

Ordonner les éléments d'un tableau en ordre croissant ou décroissant.



Principe

1. Pour chaque élément d'indice `i` allant de 0 à n-2
2. Pour chaque élément d'indice `j` allant de 1 à n-1
3. Si t[j] < t[j-1] ⇒ permuter(t[j], t[j-1])

Travail demandé

Ecrire l'algorithme de la procédure `TriBulles(n, t)` qui ordonne les éléments de `t` en ordre décroissant.

Solution - Nouveau régime

Algorithme

```
Procédure TriBilles(n: entier; @t: tab)
Début
  Pour i de 0 à n-2 faire
    Pour j de 1 à n-1 faire
      Si t[j] < t[j-1] Alors
        aux ← t[j]
        t[j] ← t[j-1]
        t[j-1] ← aux
      Fin Si
    Fin Pour
  Fin Pour
Fin
```

T.D.O.L

Objet	Type/Nature
i, j, aux	entier

Solution - Ancien régime

Algorithme

```
Procédure TriBulles(n: entier; var t: tab)
Début
  Pour i de 1 à n-1 faire
    Pour j de 2 à n faire
      Si t[j] < t[j-1] Alors
        aux ← t[j]
        t[j] ← t[j-1]
        t[j-1] ← aux
      Fin Si
    Fin Pour
  Fin Pour
Fin
```

T.D.O.L

Objet	Type/Nature
i, j, aux	entier