Etudes de complexité

Sandrine Vial
sandrine.vial@uvsq.fr

Janvier 2018

Structures Linéaires

Eléments d'un même type stockés dans :

- **un tableau**
- **une liste**

Deux cas possibles :

- Eléments triés (l'ordre doit être maintenu)
- L'ordre n'a aucune importance.

Opérations sur les structures linéaires

- Insérer un nouvel élément
- Supprimer un élément
- Rechercher un élément
- Afficher l'ensemble des éléments
- Concaténer deux ensembles d'élements
- ...

Définition des structures

Un tableau

```
Enregistrement Tab {
    T[NMAX] : entier;
    Fin     : entier;
}
```

Une liste

```
Enregistrement Elément {
   Val      : entier;
   Suivant  : ↑ Elément;
}
```

Recherche et Insertion

1. Structures non triées
    1. Recherche et insertion dans un tableau
    2. Recherche et insertion dans une liste
2. Structures triées
    1. Recherche et insertion dans un tableau
    2. Recherche et insertion dans une liste

Tableau non trié : Recherche

Algorithme 1 Recherche dans un tableau non trié

Recherche(S : Tab, x : entier) : booléen
▷ *Entrées : S (un tableau), x (élément recherché)*
▷ *Sortie : vrai si l'élément x a été trouvé dans le tableau S, faux sinon.*

```
Debut
▷ Variable Locale
    i : entier ;
pour i de 1 à S.Fin  faire
    si (S.T[i] = x)
        retourner  vrai ;
    fin si
fin pour
retourner  faux ;
Fin
```

Tableau non trié : Recherche

- **Opération fondamentale :** comparaison
- **A chaque itération :**
  - 1 comparaison (`Si ... Fin Si`)
  - 1 comparaison (`Pour ... Fin Pour`)
- **Nombre d'itérations maximum :** nombre d'éléments du tableau
- **Complexité :** Si n est le nombre d'éléments du tableau $O(n)$.

Tableau non trié : Insertion

Algorithme 2 Insertion dans un tableau non trié

Insertion(S : Tab, x : entier) : booléen

▷ ***Entrées : S (un tableau), x (élément recherché)***

▷ ***Sortie : le tableau S dans lequel x a été inséré.***

Debut

S.Fin ← S.Fin + 1 ;

S.T[S.Fin] ← x ;

Fin

Tableau non trié : Insertion

- **Opération fondamentale :** affectation
- **Nombre d'opérations fondamentales** : 2 affectations.
- **Complexité :** $O(1)$ (Temps constant).

Liste non triée : Recherche

Algorithme 3 Recherche dans une liste non triée

Recherche(L : ↑ Elément, x : entier) : booléen
▷ ***Entrées : L (tête de la liste), x (élément recherché)***
▷ ***Sortie : vrai si l'élément x a été trouvé dans la liste L, faux sinon.***

```
Debut
    si (L = NIL)
        retourner faux ;
    sinon si (L.Val = x)
        retourner vrai ;
    sinon
        retourner Recherche(L.Suivant,x) ;
    fin si
Fin
```

Liste non triée : Recherche

Algorithme 4 Recherche dans une liste non triée

```
Recherche(L : ↑ Elément, x : entier) : booléen
▷ Entrées : L (tête de la liste), x (élément recherché)
▷ Sortie : vrai si l'élément x a été trouvé dans la liste L, faux sinon.
▷ Variable Locale
    p : ↑ Element ;
Debut
    p ← L ;
    tant que (p ≠ NIL) faire
        si (p.Val ≠ x)
            p ← p.Suivant ;
        sinon
            retourner vrai ;
        fin si
    fin tant que
    retourner faux ;
Fin
```

Liste non triée : Recherche

- **Opération fondamentale :** comparaison
- **A chaque itération :**
  - 1 comparaison (`Si ... Fin Si`)
  - 1 comparaison (`Tant que ... Fin Tant Que`)
- **Nombre d'itérations maximum :** au pire le nombre d'éléments de la liste
- **Complexité :** Si n est le nombre d'éléments de la liste $O(n)$.

Liste non triée : Insertion

3 situations possibles

1. **en tête de liste**
2. en milieu de liste
3. en fin de liste

Liste non triée : Insertion

Algorithme 5 **Insertion dans une liste non triée**

Insertion(L : ↑ Element, x : entier) : booléen

▷ ***Entrées : L (tete de liste), x (élément recherché)***

▷ ***Sortie : la liste L dans laquelle x a été inséré.***

▷ ***Variable Locale***

```
    p : ↑ Element ;
Debut
    p.Val ← x ;
    p.Suivant ← L ;
    L ← p ;
Fin
```

Liste non triée : Insertion

- **Opération fondamentale :** affectation
- **Nombre d'opérations fondamentales** : 3 affectations.
- **Complexité :** $O(1)$ (Temps constant).

Liste non triée : Insertion

Algorithme 6 Insertion dans un tableau trié

Insertion(S : Tab, x : entier) : booléen

- ▷ *Entrées : S (un tableau), x (élément recherché)*
- ▷ *Sortie : le tableau S dans lequel x a été inséré.*
- ▷ *Pré-condition : le tableau S trié par ordre croissant.*
- ▷ *Variable Locale*

  i,k :entiers ;

```
Debut
    si (S.Fin == -1)
        S.Fin ← 0 ;
        S.T[S.Fin] ← x ;
    sinon
        i ← 0 ;
        tant que (i < S.Fin et
        S.T[i] < x)
            i ← i + 1 ;
        fin tant que
        si (i = S.Fin et S.T[i] <
        x)
            k ← S.Fin + 1 ;
        sinon
            k ← i ;
        fin si
        pour i de S.Fin + 1 à k
        en décroissant faire
            S.T[i] ← S.T[i-1] ;
        fin pour
        S.T[k] ← x ;
        S.Fin ← S.Fin + 1 ;
    fin si
Fin
```

Tableau trié : insertion

- **Opération fondamentale :** affectation
- **Recherche de la bonne position :** k affectations
- **Décaler à droite :** $n - k$ affectations
- **Insérer élément :** 1 affectation
- **Total :** $n + 2$ affectations
- **Complexité :** **O(n)** si n est le nombre d'éléments du tableau.

Tableau trié : recherche

1. Première idée :
   - On compare l'élément recherché à tous les éléments du tableau comme on l'a fait pour un tableau non trié.
   - Problème : on ne tient pas compte de l'ordre des éléments.
2. Deuxième idée :
   - Recherche dichotomique
   - Utilisation du fait que les éléments sont triés.

Tableau trié : recherche

- Soit M l'élément du milieu du tableau.
  - Si élément = M on a trouvé.
  - Si élément < M, l'élément est dans la première moitié du tableau.
  - Si élément > M, l'élément est dans la seconde moitié du tableau.
- Fonction récursive.

Tableau trié : recherche

Algorithme 7 Recherche dichotomique

Recherche(x : entier, S : tableau, g : entier, d : entier) : booléen

▷ ***Entrées : x (élément recherché), S (espace de recherche), g (indice de gauche), d (indice de droite)***

▷ ***Sortie : vrai si l'élément x a été trouvé dans le tableau S entre les indices g et d, faux sinon.***

▷ ***Pré-conditions : g et d sont des indices valides du tableau S et S est trié par ordre croissant.***

▷ ***Variable Locale***

```
    m : entier ;
 Debut
     si (g < d)
         m ← ⌊(g+d)/2⌋ ;
          si (x = S.T[m])
               retourner  vrai ;
          sinon  si (x < S.T[m])
               retourner (Recherche(x,S,g,m-1)) ;
          sinon
               retourner (Recherche(x,S,m+1,d)) ;
          fin si
     sinon
          retourner  faux ;
```

Tableau trié : recherche

- **Opération fondamentale :** comparaison
- ***A chaque appel récursif, on diminue l'espace de recherche par 2*** et on fait au pire 2 comparaisons
- **Complexité :** Au pire on fera donc $\mathbf{O}(\log_2 \mathbf{n})$ appels et la complexité est donc en $\mathbf{O}(\log_2 \mathbf{n})$.

Liste triée : recherche

Améliorations par rapport à une liste non triée

- Très peu ...
- Arrêt de la recherche est plus rapide.

Liste triée : recherche

Algorithme 8 Recherche dans une liste chaînée

```
Recherche(L : ↑ Element, x : entier) : booléen
▷ Entrées : x (élément recherché), L (tête de liste)
▷ Sortie : vrai si l'élément x a été trouvé dans la liste L, faux sinon.
▷ Pré-condition : La liste L est triée par ordre croissant
▷ Variable Locale
    p : ↑ Element ;
Debut
    p ← L ;
     tant que (p <> NIL)  faire
          si (p.Val < x)
               p ← p.Suivant ;
          sinon  si (p.Val = x)
               retourner  vrai ;
          sinon
               retourner  faux ;
          fin si
     fin tant que
Fin
```

Liste triée : recherche

- **Opération fondamentale :** comparaison
- **A chaque itération :**
  - Une comparaison (`Si ... Fin Si`)
  - Une comparaison (`Tant que ... Fin Tant Que`)
- **Nombre d'itérations :**au pire le nombre d'éléments de la liste.
- **Complexité :**si n est le nombre d'éléments de la liste : $O(n)$.

Liste triée : Insertion

Algorithme 9 Insertion dans une liste chaînée

Insérer(L : ↑ Element, x : entier)

▷ ***Entrées : x (élément à insérer), L (tête de liste)***

▷ ***Sortie : la liste L dans laquelle l'élément x a été inséré à sa place..***

▷ ***Pré-condition : La liste L est triée par ordre croissant***

▷ ***Variable Locale***

```
    p : ↑ Element ;
Debut
    si (L = NIL ou L.Val >= x)
        p.Val ← x ;
        p.Suivant ← L ;
        L ← p ;
    sinon
        L.Suivant ← Insérer(L.Suivant,x) ;
    fin si
Fin
```

Liste triée : insertion

- **Opération fondamentale :** comparaison
- **A chaque itération :** 2 comparaisons
- **Nombre d'itérations :** au pire il faut parcourir tous les éléments de la liste.
- **Complexité :** si n est le nombre d'éléments de la liste : $O(n)$.

Résumé

Complexité de l'insertion

	Eléments triés	Eléments non triés
Tableau	$O(n)$	$O(1)$
Liste	$O(n)$	$O(1)$

Complexité de la recherche

	Eléments triés	Eléments non triés
Tableau	$O(\log_2 n)$	$O(n)$
Liste	$O(n)$	$O(n)$