Etudes de complexité

Sandrine Vial
sandrine.vial@uvsq.fr

Septembre 2020

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;
}
```

Recherche et Insertion

1. Tableau non trié
    1. Recherche
    2. Insertion
2. Tableau trié
    1. Recherche
    2. Insertion

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)
▷ Entrées : S (un tableau), x (élément à insérer)
▷ 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).

Tableau trié : Insertion

Algorithme 3 Insertion dans un tableau trié

Insertion(S : Tab, x : entier)

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

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

▷ ***Pré-condition : le tableau S trié par ordre croissant.***

▷ ***Variables Locales***

i,k :entiers ;

Debut
si (S.Fin = 0)
S.Fin ← 1 ;
S.T[S.Fin] ← x ;
sinon
i ← 1 ;
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 :** 2 affectations
- **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.
   - Avantage : on s'arrête dès que l'on tombe sur un élément plus grand que la valeur recherchée.
2. Deuxième idée :
   - Recherche dichotomique
   - Utilisation du fait que les éléments sont triés.

Tableau trié : recherche dichotomique

- 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 dichotomique

Algorithme 4 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 ;
    fin si
Fin
```

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 $O(\log_2 n)$ appels et la complexité est donc en $O(\log_2 n)$.

Résumé

Complexité de l'insertion

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

Complexité de la recherche

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