Haute École de Bruxelles
École Supérieure d'Informatique

TD Tri

Résumé

Dans ce chapitre nous voyons quelques algorithmes simples pour trier un ensemble d'informations : recherche des maxima, tri par insertion et tri bulle dans un tableau. Des algorithmes plus efficaces seront vus en deuxième année.

1 Motivation

La recherche efficace d'information implique un tri préalable de celle-ci. En effet, si les données ne sont pas classées ou triées, le seul algorithme possible reviendrait à parcourir entièrement l'ensemble des informations. Pour exemple, il suffit d'imaginer un dictionnaire dans lequel les mots seraient mélangés de façon aléatoire au lieu d'être classés par ordre alphabétique. Pour trouver le moindre mot dans ce dictionnaire, il faudrait à chaque fois le parcourir entièrement ! Il est clair que le classement préalable (ordre alphabétique) accélère grandement la recherche.

Ainsi, recherche et tri sont étroitement liés, et la façon dont les informations sont triées conditionne bien entendu la façon de rechercher l'information (cf. algorithme de recherche dichotomique). Pour exemple, prenons cette fois-ci un dictionnaire des mots croisés dans lequel les mots sont d'abord regroupés selon leur longueur et ensuite par ordre alphabétique. La façon de rechercher un mot dans ce dictionnaire est bien sûr différente de la recherche dans un dictionnaire usuel.

Le problème central est donc le tri des informations. Celui-ci a pour but d'organiser un ensemble d'informations qui ne l'est pas à priori. On peut distinguer trois grands cas de figure :

données «brutes »

D'abord les situations impliquant le classement total d'un ensemble de données «brutes », c'est-à-dire complètement désordonnées. Prenons pour exemple les feuilles récoltées en vrac à l'issue d'un examen ; il y a peu de chances que celles-ci soient remises à l'examinateur de manière ordonnée ; celui-ci devra donc procéder au tri de l'ensemble des copies, par exemple par ordre alphabétique des noms des étudiants, ou par numéro de groupe etc.

ajout dans un ensemble trié

Ensuite les situations où on s'arrange pour ne jamais devoir trier la totalité des éléments d'un ensemble, qui resterait cependant à tout moment ordonné. Imaginons le cas d'une bibliothèque dont les livres sont rangés par ordre alphabétique des auteurs : à l'achat d'un nouveau livre, ou au retour de prêt d'un livre, celui-ci est immédiatement rangé à la bonne place. Ainsi, l'ordre global de la bibliothèque est maintenu par la répétition d'une seule opération élémentaire consistant à insérer à la bonne place un livre parmi la collection. C'est la situation que nous considérerions dans le cas d'une structure où les éléments sont ordonnés.

re-trier

Enfin, les situations qui consistent à devoir re-trier des données préalablement ordonnées sur un autre critère. Prenons l'exemple d'un paquet de copies d'examen déjà triées sur l'ordre alphabétique des noms des étudiants, et qu'on veut re-trier cette fois-ci sur les numéros de groupe. Il est clair qu'une méthode efficace veillera à conserver l'ordre alphabétique déjà présent dans la première situation afin que les copies apparaissent dans cet ordre dans chacun des groupes.

Le dernier cas illustre un classement sur une clé complexe (ou composée) impliquant la comparaison de plusieurs champs d'une même structure : le premier classement se fait sur le numéro de groupe, et à numéro de groupe égal, l'ordre se départage sur le nom de l'étudiant. On dira de cet ensemble qu'il est classé en **majeur** sur le numéro de groupe et en **mineur** sur le nom d'étudiant.

Notons que certains tris sont dits **stables** parce qu'en cas de tri sur une nouvelle clé, l'ordre de la clé précédente est préservé pour des valeurs identiques de la nouvelle clé, ce qui évite de faire des comparaisons sur les deux champs à la fois. Les méthodes nommées tri par insertion, tri bulle et tri par recherche de minima successifs (que nous allons aborder) sont stables.

1.1 Tri par insertion

Cette méthode de tri repose sur le principe d'insertion de valeurs dans un tableau ordonné.

Le tableau à trier sera à chaque étape subdivisé en deux sous-tableaux : le premier cadré à gauche contiendra des éléments déjà ordonnés, et le second, cadré à droite, ceux qu'il reste à insérer dans le sous-tableau trié. Celui-ci verra sa taille s'accroitre au fur et à mesure des insertions, tandis que celle du sous-tableau des éléments non triés diminuera progressivement. Au départ de l'algorithme, le sous-tableau trié est le premier élément du tableau. Comme il ne possède qu'un seul élément, ce sous-tableau est donc bien ordonné ! Chaque étape consiste ensuite à prendre le premier élément du sous-tableau non trié et à l'insérer à la bonne place dans le sous-tableau trié.

```
/// Trie le tableau reçu en paramètre (via un tri par insertion).
module triInsertion(tab↓↑ : tableau de n entiers)
    i, j, valAInsérer : entiers
    pour i de 1 à n-1 faire
      valAInsérer ← tab[i]
      // recherche de l'endroit où insérer valAInsérer dans le
      // sous-tableau trié et décalage simultané des éléments
```

```
      j ← i-1
      tant que j ≥ 0 ET valAInsérer < tab[j] faire
        tab[j+1] ← tab[j]
        j ← j-1
      fin tant que
      tab[j+1] ← valAInsérer
    fin pour
fin module
```

1.2 Tri par sélection des minima successifs

Dans ce tri, on recherche à chaque étape la plus petite valeur de l'ensemble non encore trié et on peut la placer immédiatement à sa position définitive.

```
// Trie le tableau reçu en paramètre (via un tri par sélection des minima successifs
module triSélectionMinimaSuccessifs(tab↓↑ : tableau de n entiers)
    i, indiceMin : entier
    pour i de 0 à n - 2 faire // i correspond à l'étape de l'algorithme
      indiceMin ← positionMin( tab, i, n-1 )
      swap( tab[i], tab[indiceMin] )
    fin pour
fin module


// Échange le contenu de 2 variables.
module swap(a↓↑, b↓↑ : entiers)
    aux : entiers
    aux ← a
    a ← b
    b ← aux
fin module


// Retourne l'indice du minimum entre les indices début et fin du tableau reçu.
module positionMin(tab↓ : tableau de n entiers, début, fin : entiers) → entier
    indiceMin, i : entiers
    indiceMin ← début
    pour i de début+1 à fin faire
      si tab[i] < tab[indiceMin] alors
        indiceMin ← i
      fin si
    fin pour
    retourner indiceMin
```

```
fin module
```

1.3 Tri bulle

Il s'agit d'un tri par permutations ayant pour but d'amener à chaque étape à la «surface »du sous-tableau non trié (on entend par là l'élément d'indice minimum) la valeur la plus petite, appelée la bulle. La caractéristique de cette méthode est que les comparaisons ne se font qu'entre éléments consécutifs du tableau.

```
// Trie le tableau reçu en paramètre (via un tri bulle).
module triBulle(tab↓↑ : tableau de n entiers)
    indiceBulle, i : entiers
    pour indiceBulle de 1 à n faire
      pour i de n - 2 à indiceBulle par - 1 faire
        si tab[i] > tab[i + 1] alors
          swap( tab[i], tab[i + 1] )
        fin si
      fin pour
    fin pour
fin module
```

```
// Échange le contenu de 2 variables.
module swap(a↓↑, b↓↑ : entiers)
    aux : entiers
    aux ← a
    a ← b
    b ← aux
fin module
```

1.4 Recherche dichotomique

La recherche dichotomique a pour essence de réduire à chaque étape la taille de l'ensemble de recherche de moitié, jusqu'à ce qu'il ne reste qu'un seul élément dont la valeur devrait être celle recherchée, sauf bien entendu en cas d'inexistence de cette valeur dans l'ensemble de départ.

Soit val la valeur recherchée dans une zone délimitée par les indices indiceGauche et indiceDroit. On commence par déterminer l'élément médian,

c'est-à-dire celui qui se trouve «au milieu »de la zone de recherche ; son indice sera déterminé par la formule `indiceMédian ← (indiceGauche + indiceDroit) DIV 2`

N.B. : cet élément médian n'est pas tout à fait au milieu dans le cas d'une zone contenant un nombre pair d'éléments. On compare alors val avec la valeur de cet élément médian ; il est possible qu'on ait trouvé la valeur cherchée ; sinon, on partage la zone de recherche en deux parties : une qui ne contient certainement pas la valeur cherchée et une qui pourrait la contenir. C'est cette deuxième partie qui devient la nouvelle zone de recherche. On réitère le processus jusqu'à ce que la valeur cherchée soit trouvée ou que la zone de recherche soit réduite à sa plus simple expression, c'est-à-dire un seul élément.

```
module rechercheDichotomique(tab↓↑ : tableau de n T, valeur↓ : T, pos↑ : entier) →
    indiceDroit, indiceGauche, indiceMédian : entiers
    candidat : T
    trouvé : booléen
    indiceGauche ← 0
    indiceDroit ← n-1
    trouvé ← faux
    tant que NON trouvé ET indiceGauche ≤ indiceDroit faire
      indiceMédian ← (indiceGauche + indiceDroit) DIV 2
      candidat ← tab[indiceMédian]
      si candidat = valeur alors
        trouvé ← vrai
      sinon si candidat < valeur alors
        indiceGauche ← indiceMédian + 1 // on garde la partie droite
      sinon
        candidat > valeur: indiceDroit ← indiceMédian - 1 // on garde la partie gau
      fin si
    fin tant que

    si trouvé alors
      pos ← indiceMédian
    sinon
      pos ← indiceGauche
      // dans le cas où la valeur n'est pas trouvée,
      // on vérifiera que indiceGauche donne la valeur où elle pourrait être insérée
    fin si
    retourner trouvé
fin module
```

2 Exercices

Maintenant, mettons tout ça en pratique.

2.1 En Java...

N'oubliez pas nos quelques conseils pour vous guider dans la résolution de tels problèmes :

- il convient d'abord de bien comprendre le problème posé ; assurez-vous qu'il est parfaitement spécifié ;
- résolvez le problème via quelques exemples précis ;
- mettez en évidence les variables **« données »**, les variables **« résultats »** et les variables de travail ;
- n'hésitez pas à faire une ébauche de résolution en français avant d'élaborer l'algorithme définitif pseudo-codé ;
- déclarez ensuite les variables (et leur type) qui interviennent dans chaque module ; les noms des variables risquant de ne pas être suffisamment explicites.
- Écrivez la partie algorithmique **AVANT** de vous lancer dans la programmation en Java.
- Demandez-vous si vous avez besoin de parcourir tout le tableau ou de sortir prématurément (si on a trouvé ce qu'on cherche par exemple).
- Pour la partie Java, dessinez l'arborescence des fichiers.
- **Écrivez le plan de tests en écrivant l'algorithme. Codez les tests après avoir écrit le code Java.**

Écrivez en Java les algorithmes de tri vus ici en affichant le tableau à chaque étape.

Sélection des maxima

Développez un algorithme similaire consistant à trier un tableau par ordre croissant par sélection des maxima successifs. Le sous-tableau trié apparaitra donc à droite du tableau, et les maxima sélectionnés seront à chaque étape positionnés à droite du sous-tableau non trié.

Maxima et minima

Développez un algorithme combinant les deux recherches. À chaque étape, on sélectionne donc le minimum et le maximum du sous-tableau restant à trier et on les positionnera à l'endroit ad hoc. Cette méthode apporte-t-elle une amélioration en temps ou en simplicité aux deux algorithmes de base ?

Amélioration du tri bulle

Écrivez une amélioration du tri bulle consistant à mémoriser à chaque étape l'indice de la dernière permutation, celui-ci délimitant en fait la véritable taille du sous-tableau trié à l'issue d'une étape. En lieu et place de la boucle pour n'incrémentant l'indice bulle que de 1 à la fois, vous écrirez une boucle `tant que` à l'issue de laquelle `indiceBulle` prendra la valeur de l'*indice de dernière permutation + 1*.

Tri shaker

L'amélioration précédente est issue de l'observation du sous-tableau déjà trié en début du tableau initial. On peut de même étudier la possibilité d'avoir un sous-tableau, trié égale- ment, mais cadré à droite dans le tableau à trier. Cette symétrie suggère une amélioration supplémentaire qui consiste à changer de sens à la fin d'un parcours pour entamer le parcours suivant. Lorsqu'on change de sens, on amènera l'élément le plus grand (qu'on peut nommer le «plomb ») au fond du tableau non trié. L'association des deux méthodes donne ce qu'on appelle le tri shaker, dont le but est de restreindre le sous-tableau non trié en augmentant sa borne inférieure et en diminuant sa borne supérieure. Écrivez l'algorithme qui réalise cette méthode de tri.