

R1.01

INITIATION AU DÉVELOPPEMENT

Cours 7, partie 3 : Vecteurs

✓ Algorithmes outillés

Hervé Blanchon & Anne Lejeune

Université Grenoble Alpes

IUT 2 – Département Informatique

Sommaire

 Objectif et exemple

 Outillage d'une procédure

 Outillage d'une fonction

 Traces


OBJECTIF

Nombre d'exécutions d'une instruction ?

Comptage théorique

 dans la SAÉ1.02 : comparaison d'approches algorithmiques

Comptage pratique

 il faut « outiller » la fonction ou la procédure dans laquelle on veut faire le comptage


 Outiller une procédure

 elle va devenir une fonction qui retourne la valeur d'un compteur

 Outiller une fonction

 elle va devoir retourner deux résultats...

 le résultat de la fonction proprement dit

 la valeur d'un compteur

 ...au moyen d'une classe `PaireResultatCompteur<R>`

Exemples

(traités, *non traités* ici)


 Combien de comparaisons entre deux éléments du vecteur à trier dans les procédures de tri ?

 tri par sélection

 tri par insertion

 *tri à bulle*

 *tri à bulle amélioré*

 Combien de comparaisons avec un élément du vecteur dans les fonctions de recherche dans un vecteur quelconque ou trié ?

 recherche séquentielle dans un vecteur quelconque

 recherche séquentielle dans un vecteur trié

 *recherche dichotomique dans un vecteur trié*

OUTILLAGE D'UNE PROCÉDURE

Exemple 1 : tri par sélection

Situation facile → comparaisons hors d'une condition d'itération



Code
original :

```
1 private static void triSelection(ArrayList<Integer> v) {  
2     int i = 0;  
3     while (i < v.size()-1) {  
4         int indMin = i;  
5         int j = i + 1;  
6         while (j < v.size()) {  
7             if (v.get(j) < v.get(indMin)) {  
8                 indMin = j;  
9             }  
10            j = j + 1;  
11        }  
12        if (indMin != i) {  
13            int temporaire = v.get(i);  
14            v.set(i, v.get(indMin));  
15            v.set(indMin, temporaire);  
16        }  
17        i = i + 1;  
18    }  
19 }
```



Compter le nombre de comparaisons entre éléments du vecteur **v** revient à compter le nombre d'exécutions de la ligne 7



c'est facile, exactement une comparaison par itération, ligne 6

Mise en œuvre du comptage (procédure)

 La procédure va être transformée en fonction :

```
private static int triSelectionAvecNbComparaisons(ArrayList<Integer> v) {  
    ...  
}
```

 Déclarer et initialiser un compteur dans la zone de déclaration des variables

```
int nbComparaisons = 0;
```

Mise en œuvre du comptage (procédure)

 Incrémenter le compteur dans le bloc d'instructions `while` concerné

 `while (j < v.size())`

```
while (j < v.size()) {  
    if (v.get(j) < v.get(indMin)) {  
        indMin = j;  
    }  
    // on a fait une comparaison dans le bloc du while  
    nbComparaisons = nbComparaisons + 1;  
  
    j = j + 1;  
}
```

```
private static int triSelectionAvecNbComparaisons(ArrayList<Integer> v) {
    int i = 0;
    int nbComparaisons = 0;

    while (i < v.size()-1) {
        int indMin = i;
        int j = i + 1;
        while (j < v.size()) {
            if (v.get(j) < v.get(indMin)) {
                indMin = j;
            }
            // on a fait une comparaison dans le bloc du while
            nbComparaisons = nbComparaisons + 1;
            j = j + 1;
        }
        if (indMin != i) {
            int temporaire = v.get(i);
            v.set(i, v.get(indMin));
            v.set(indMin, temporaire);
        }
        i = i + 1;
    }
    return nbComparaisons; // retourner le nombre de comparaisons
}
```

Exemple 2 : tri par insertion

Situation plus difficile → comparaison à l'intérieur d'une condition d'itération



Code
original

```
1 private static void triInsertion(ArrayList<Integer> v) {  
2     int j;  
3     int valeurAPlacer;  
4     int i = 1;  
5     while (i < v.size()) {  
6         j = i;  
7         valeurAPlacer = v.get(i);  
8         while (j > 0 && v.get(j-1) > valeurAPlacer) {  
9             v.set(j, v.get(j-1));  
10            j = j - 1;  
11        }  
12        v.set(j, valeurAPlacer);  
13        i = i + 1;  
14    }  
15 }
```



La comparaison à lieu ligne 8, dans une condition d'itération



il faut donc ajouter une comparaison pour chaque exécution du bloc de l'itération de la ligne 8



il faut peut-être ajouter une comparaison après l'instruction d'itération (à partir de la ligne 12)



en effet : si on est sortie de l'itération alors que $j > 0$ c'est que $v.get(j-1) \leq valeurAPlacer$ (la comparaison $v.get(j-1) > valeurAPlacer$ fausse fait sortir de l'itération)

Mise en œuvre du comptage (procédure)


 La procédure va être transformée en fonction :

```
private static int triInsertionAvecNbComparaisons(ArrayList<Integer> v) {  
    ...  
}
```

 Déclarer et initialiser un compteur dans la zone de déclaration des variables

```
int nbComparaisons = 0;
```


Mise en œuvre du comptage (procédure)

 Incrémenter le compteur dans le bloc d'instructions du bloc du while concerné

 while (j > 0 && v.get(j-1) > valeurAPlacer)

```
while (j > 0 && v.get(j-1) > valeurAPlacer) {  
    v.set(j, v.get(j-1));  
    j = j - 1;  
    // une comparaison pour rentrer dans le bloc du while  
    nbComparaisons = nbComparaisons + 1;  
}
```

```
// si sortie de l'itération alors que j > 0 c'est que v.get(j-1) <= valeurAPlacer  
// alors une comparaison de plus (la comparaison fait sortir de l'itération)  
if (j > 0) {  
    nbComparaisons = nbComparaisons + 1;  
}
```










 Si la comparaison rend fausse la condition d'itération alors une comparaison de plus

```
private static int triInsertionAvecNbComparaisons(ArrayList<Integer> v) {
    int j;
    int valeurAPlacer;
    int i = 1;
    int nbComparaisons = 0;

    while (i < v.size()) {
        j = i;
        valeurAPlacer = v.get(i);
        while (j > 0 && v.get(j-1) > valeurAPlacer) {
            v.set(j, v.get(j-1));
            j = j - 1;
            // une comparaison pour rentrer dans le bloc
            nbComparaisons = nbComparaisons + 1;
        }
        // si sortie de l'itération alors que j > 0 c'est que v.get(j-1) <= valeurAPlacer
        // alors une comparaison de plus (la comparaison fait sortir de l'itération)
        if (j > 0) {
            nbComparaisons = nbComparaisons + 1;
        }
        v.set(j, valeurAPlacer);
        i = i + 1;
    }
    return nbComparaisons; // retourner le nombre de comparaisons
}
```


OUTILLAGE D'UNE FONCTION

Remarques

-  Une fonction retourne déjà un résultat
-  Comment faire pour retourner en plus une valeur mesurée ?
 -  compteur du nombre d'exécutions d'une partie du code à observer
-  On propose une classe générique immuable :
 -  `PaireResultatCompteur<R>`
 -  R s'appelle un paramètre de type de la classe
 -  cette classe à **deux attributs constants**
 -  `resultat` (le résultat « naturel » de la fonction à outiller)
 -  `compteur` (le nombre d'exécution de la partie du code à observer)

```
/**
 * Classe générique pour outiller une fonction
 * Elle propose uniquement un constructeur et 2 getters
 * @param <R> : R est le type du résultat de la fonction outillée
 */
public class PaireResultatCompteur<R> {

    private final R resultat; // résultat constant de la fonction outillée
    private final int compteur; // compteur constant du code observé

    public PaireResultatCompteur(R resultat, int compteur) {
        this.resultat = resultat;
        this.compteur = compteur;
    }

    public R getResultat() {
        return resultat;
    }

    public int getCompteur() {
        return compteur;
    }
}
```

En pratique : modification de l'entête



 Soit à outiller la fonction suivante :

```
private static boolean  
estEntierPresentNonTrie(ArrayList<Integer> v, int val) {...}
```

 Son entête outillé pour compter le nombre de comparaisons devient :

```
private static PaireResultatCompteur<Boolean>  
estEntierPresentNonTrieAvecNbComparaisons(ArrayList<Integer> v,  
                                             int val) {...}
```

Remarques


-  le paramètre de type de la classe PaireResultatCompteur est obligatoirement un nom de classe
(comme pour `ArrayList<E>` – `E` pour `Element`)
-  si le résultat de la fonction est un type primitif (ex. `boolean`) on utilise la classe enveloppe associée (ex. `Boolean`)

En pratique : modification du corps


 Comme pour une procédure

 il faut déclarer et initialiser un compteur entier

 `int nbComparaison = 0; // si on compte des comparaisons`

 il faut mettre à jour le compteur pour tenir compte de chaque exécution de la/des instruction(s) à observer

 Au lieu de retourner le résultat « naturel » :

 il faut construire et retourner un objet de type `PaireResultatCompteur<R>` construite avec le résultat « naturel » et le compteur

Exemple : recherche associative non triée

Situation plus difficile → comparaison à l'intérieur d'une condition d'itération



Code original

```
1 private static boolean
  estEntierPresentNonTrie(ArrayList<Integer> v, int val) {
2     int i = 0;
3     while (i < v.size() && v.get(i) != val) {
4         i = i + 1;
5     }
6     // instruction pour produire le résultat naturel
7     return i < v.size();
8 }
```



La comparaison à lieu ligne 3, dans une condition d'itération



il faut donc ajouter une comparaison pour chaque exécution du bloc de l'itération de la ligne 3



il faut peut-être ajouter une comparaison après l'instruction d'itération (à partir de la ligne 6)



en effet : si on est sortie de l'itération alors que `i < v.size()` c'est que `v.get(i) == val` (la comparaison `v.get(i) != val` fausse fait sortir de l'itération)

Mise en œuvre du comptage (fonction)


 Le résultat de la fonction est modifié :


```
private static PaireResultatCompteur<Boolean>  
estEntierPresentNonTrieAvecNbComparaisons(ArrayList<Integer> v,  
                                             int val) {...}
```

 Déclarer et initialiser un compteur dans la zone de déclaration des variables

```
int nbComparaisons = 0;
```



Mise en œuvre du comptage (fonction)

 Incrémenter le compteur dans le bloc d'instructions du bloc du while concerné

 `while (i > v.size() && v.get(i) != val)`

`while (i < v.size() && v.get(i) != val) {
 i = i + 1;
 // une comparaison pour rentrer dans le bloc du while
 nbComparaisons = nbComparaisons + 1;
}`

`// si sortie de l'itération alors que j < v.size() c'est que v.get(i) == val
// alors une comparaison de plus (la comparaison fait sortir de l'itération)
if (i < v.size()) {
 nbComparaisons = nbComparaisons + 1;
}`

 Si la comparaison rend fausse la condition d'itération alors une comparaison de plus

```
private static PaireResultatCompteur<Boolean>
estEntierPresentNonTrieAvecNbComparaisons(ArrayList<Integer> v, int val) {

    int i = 0;
    int nbComparaisons = 0;    // déclaration et initialisation du compteur

    while (i < v.size() && v.get(i) != val) {
        i = i + 1;
        // une comparaison pour rentrer dans le bloc du while
        nbComparaisons = nbComparaisons + 1;
    }

    // si sortie de l'itération alors que j < v.size() et alors v.get(j) == val
    // alors une comparaison de plus (la comparaison fait sortir de l'itération)
    if (i < v.size()) {
        nbComparaisons = nbComparaisons + 1;
    }

    // construire et retourner le résultat, une PaireResultatCompteur,
    // construite avec comme premier paramètre, le résultat naturel de la fonction
    //             comme deuxième paramètre, le nombre de comparaisons
    return new PaireResultatCompteur<>(i < v.size(), nbComparaisons);
}
```

Exemple : recherche associative triée

Situation plus difficile → comparaison à l'intérieur d'une condition d'itération



Code original

```
1 private static boolean
  estEntierPresentNonTrie(ArrayList<Integer> v, int val) {
2     int i = 0;
3     while (i < v.size() && v.get(i) < val) {
4         i = i + 1;
5     }
6     // instruction pour produire le résultat naturel
7     return i < v.size() && v.get(i) == val;
8 }
```



La comparaison à lieu ligne 3, dans une condition d'itération



il faut donc ajouter une comparaison pour chaque exécution du bloc de l'itération de la ligne 3



il faut peut-être ajouter une comparaison après l'instruction d'itération (à partir de la ligne 6)



en effet : si on est sortie de l'itération alors que $i < v.size()$ c'est que $v.get(i) \geq val$ (la comparaison $v.get(i) < val$ fausse fait sortir de l'itération)

Mise en œuvre du comptage (fonction)


 Le résultat de la fonction est modifié :

```
private static PaireResultatCompteur<Boolean>  
estEntierPresentTrieAvecNbComparaisons(ArrayList<Integer> v,  
                                         int val) {...}
```

 Déclarer et initialiser un compteur dans la zone de déclaration des variables

```
int nbComparaisons = 0;
```


Mise en œuvre du comptage (fonction)

 Incrémenter le compteur dans le bloc d'instructions du bloc du while concerné

 `while (i > v.size() && v.get(i) < val)`

`while (i < v.size() && v.get(i) < val) {
 i = i + 1;
 // une comparaison pour rentrer dans le bloc du while
 nbComparaisons = nbComparaisons + 1;
}`

`// si sortie de l'itération alors que j < v.size() c'est que v.get(j) >= val
// alors une comparaison de plus (la comparaison fait sortir de l'itération)
if (i < v.size()) {
 nbComparaisons = nbComparaisons + 1;
}`

 Si la comparaison rend fausse la condition d'itération alors une comparaison de plus

```
private static PaireResultatCompteur<Boolean>
estEntierPresentTrieAvecNbComparaisons(ArrayList<Integer> v, int val) {

    int i = 0;
    int nbComparaisons = 0;    // déclaration et initialisation du compteur

    while (i < v.size() && v.get(i) < val) {
        i = i + 1;
        // une comparaison pour rentrer dans le bloc du while
        nbComparaisons = nbComparaisons + 1;
    }

    // si sortie de l'itération alors que j < v.size() c'est que v.get(j) == val
    // alors une comparaison de plus (la comparaison fait sortir de l'itération)
    if (i < v.size()) {
        nbComparaisons = nbComparaisons + 1;
    }

    // construire et retourner le résultat, une PaireResultatCompteur,
    // construite avec comme premier paramètre, le résultat naturel de la fonction
    //             comme deuxième paramètre, le nombre de comparaisons
    return new PaireResultatCompteur<>(i < v.size() && v.get(i) == val,
                                       nbComparaisons);
}
```

TRACES DES PROCÉDURES ET FONCTIONS OUTILLÉES

Code du test des procédures de tri

```
public static void main(String[] args) {
    ArrayList<Integer> unVectInt = new ArrayList<>(Arrays.asList(12, ..., 12));

    System.out.println("unVectInt à l'origine avant tri par sélection : " + unVectInt);

    int nbComparaisons = triSelectionAvecNbComparaisons(unVectInt);

    System.out.println("unVectInt après tri par sélection : " + unVectInt);
    System.out.println("-> l'algorithme a fait " + nbComparaisons + " comparaisons");

    ArrayList<Integer> unVectInt3 = new ArrayList<>(Arrays.asList(12, ..., 12));

    System.out.println("unVectInt3 à l'origine avant tri par insertion: " + unVectInt3);

    nbComparaisons = triInsertionAvecNbComparaisons(unVectInt3);

    System.out.println("unVectInt3 après tri par insertion : " + unVectInt3);
    System.out.println("-> l'algorithme a fait " + nbComparaisons + " comparaisons");
}
```

Trace du test des procédures de tri

unVectInt à l'origine avant tri par sélection :
[12, 7, 9, 14, 5, 17, 6, 8, 12]

unVectInt après tri par sélection :
[5, 6, 7, 8, 9, 12, 12, 14, 17]
-> l'algorithme a fait 36 comparaisons

unVectInt3 à l'origine avant tri par insertion:
[12, 7, 9, 14, 5, 17, 6, 8, 12]
unVectInt3 après tri par insertion :
[5, 6, 7, 8, 9, 12, 12, 14, 17]
-> l'algorithme a fait 23 comparaisons

Code du test des fonctions de recherche

```
public static void main(String[] args) {
    ArrayList<Integer> vectInteger = new ArrayList<>(Arrays.asList(45, 12, 28, 85, 10));
    System.out.println("Vecteur non trié : " + vectInteger);

    PaireResultatCompteur<Boolean> rechEntier;

    rechEntier = estEntierPresentNonTrieAvecNbComparaisons(vectInteger, 9);
    System.out.println("9 ? : " + rechEntier.getResultat() + ", nombre de comparaisons : "
        + rechEntier.getCompteur());

    ...
    rechEntier = estEntierPresentNonTrieAvecNbComparaisons(vectInteger, 90);
    System.out.println("90 ? : " + rechEntier.getResultat() + ", nombre de comparaisons : "
        + rechEntier.getCompteur());

    ArrayList<Integer> vectTrieInteger = new ArrayList<>(Arrays.asList(10, 12, 28, 45, 85));
    System.out.println("Vecteur trié : " + vectTrieInteger);

    rechEntier = estEntierPresentTrieAvecNbComparaisons(vectTrieInteger, 9);
    System.out.println("9 ? : " + rechEntier.getResultat() + ", nombre de comparaisons : "
        + rechEntier.getCompteur());

    ...
    rechEntier = estEntierPresentTrieAvecNbComparaisons(vectTrieInteger, 90);
    System.out.println("90 ? : " + rechEntier.getResultat() + ", nombre de comparaisons : "
        + rechEntier.getCompteur());

}
```

Trace du test des fonctions de recherche

```
Vecteur non trié : [45, 12, 28, 85, 10]
9 ? : false, nombre de comparaisons : 5
10 ? : true, nombre de comparaisons : 5
28 ? : true, nombre de comparaisons : 3
45 ? : true, nombre de comparaisons : 1
85 ? : true, nombre de comparaisons : 4
90 ? : false, nombre de comparaisons : 5
```

```
Vecteur trié : [10, 12, 28, 45, 85]
9 ? : false, nombre de comparaisons : 1
10 ? : true, nombre de comparaisons : 1
28 ? : true, nombre de comparaisons : 3
45 ? : true, nombre de comparaisons : 4
85 ? : true, nombre de comparaisons : 5
90 ? : false, nombre de comparaisons : 5
```