

# Cours 1

## Tri et recherche en table

[Jean-Jacques.Levy@inria.fr](mailto:Jean-Jacques.Levy@inria.fr)

<http://jeanjacqueslevy.net>

tel: 01 39 63 56 89

secrétariat de l'enseignement:

**Catherine Bensoussan**

[cb@lix.polytechnique.fr](mailto:cb@lix.polytechnique.fr)

Aile 00, LIX

tel: 01 69 33 34 67

<http://www.enseignement.polytechnique.fr/informatique/>

## Plan

1. Premier programme
2. Rappels de Java
3. Tri sélection
4. Tri par insertions
5. Recherche linéaire
6. Hachage et listes de collisions
7. Hachage à adressage ouvert

## Premier programme

Programme qui affiche une chaîne et un entier.

```
class PremierProg {
    public static void main (String args[ ]){
        System.out.println ("Bonjour les forts!");
        System.out.println ("fib(20) = " + fib(20));
    }

    static int fib (int n) {
        if (n <= 1) return 1;
        else return fib (n-1) + fib (n-2);
    }
}
```

**A mettre dans un fichier PremierProg.java, ensuite compiler**

```
javac PremierProg.java
```

**et exécuter le fichier PremierProg.class en lançant la JVM**

```
java PremierProg
```

## Types primitifs et Conversions

- déclaration  $t \ x_1 = e_1, x_2 = e_2, \dots x_n = e_n;$ 
  - Octets (8 bits) byte
  - Entiers courts (16 bits) short
  - Entiers (32 bits) int
  - Entiers longs (64 bits) long
  - Réels (32 bits) float
  - Réels longs (64 bits) double
  - Caractères (16 bits) char
  - Booléens boolean
- Conversions implicites:
  - byte  $\rightarrow$  short  $\rightarrow$  int  $\rightarrow$  long  $\rightarrow$  float  $\rightarrow$  double
  - char  $\rightarrow$  int
- Conversions explicites:  $(t) \ e$

où  $t$  est un type,  $e$  une expression et  $x$  un identificateur.

## Instructions

- $x = e;$
- $\{ inst_1 inst_2 \dots inst_n \}$  où  $inst_1, inst_2, \dots, inst_n$  sont des instructions quelconques ( $n \geq 0$ ).
- if ( $e$ ) instruction
- if ( $e$ ) instruction<sub>1</sub> else instruction<sub>2</sub>
- while ( $e$ ) instruction
- for ( $affectation_1; e; affectation_2$ ) instruction
- return  $e;$

## Tableaux

- **déclarations:** 2 notations sont tolérées. (Nous préférons la 1ère; la seconde est pour les fans C ou C++).
  - `type[ ] nom_du_tableau;`
  - `type nom_du_tableau[ ];`
- `int[ ] x = {1, 4, 9, 16, 25, 36, 49 };`
- `int[ ] x = new int [7];` (7 zéros)
- `x[i]` pour le  $i+1$ <sup>ème</sup> élément
- `x[i] = e;` pour modifier le  $i+1$ <sup>ème</sup> élément
- `x.length` est la longueur du tableau `x`
- la valeur d'un tableau est l'adresse de l'objet qui le représente.

**Exercice 1** Donner le sens de l'instruction `a = b;` quand `a` et `b` sont deux tableaux.

## Fonctions

En Java, les fonctions sont aussi appelées **méthodes** car c'est la terminologie dans les langages orientés-objets. Nous utilisons les deux appellations.

```
class PremierProg {
    public static void main (String args[ ]){
        if (args.length != 1)
            System.out.println ("Mauvais nombre d'arguments.");
        else {
            int n = Integer.parseInt (args[0]);
            System.out.println (fib(n));
        }
    }

    static int fib (int n) {
        if (n <= 1) return 1;
        else return fib (n-1) + fib (n-2);
    }
}
```

**La déclaration d'une fonction commence le type du résultat, puis vient le nom de la fonction et la déclaration des paramètres, et enfin on met le corps de la définition entre accolades.**

## Initialisation / Impression de tableaux

```
static int[] tableauAleatoire (int n) {
    int a[] = new int[n];
    for (int i = 0; i < a.length; ++i)
        a[i] = (int) (Math.random() * 100);
    return a;
}

static void imprimer (int[] a) {
    for (int i = 0; i < a.length; ++i)
        System.out.print(a[i] + " ");
    System.out.println();
}
```

**Remarque:** + est le seul **opérateur surchargé** de Java. Le sens de + est normal quand ses arguments sont numériques, mais dès qu'un de ses arguments est une chaîne de caractères, le 2ème argument est transformé en chaîne de manière naturelle et le résultat est la concaténation des deux chaînes de caractères.

## Programme complet

```
class TableauI0 {
    public static void main (String args[ ]){
        if (args.length != 1)
            System.out.println ("Mauvais nombre d'arguments.");
        else {
            int n = Integer.parseInt (args[0]);
            int[] a = tableauAleatoire (n);
            imprimer (a);
        }
    }

    static int[] tableauAleatoire (int n) {
        int a[ ] = new int[n];
        for (int i = 0; i < a.length; ++i)
            a[i] = (int) (Math.random() * 100);
        return a;
    }

    static void imprimer (int[ ] a) {
        for (int i = 0; i < a.length; ++i)
            System.out.print(a[i] + " ");
        System.out.println();
    }
}
```

## Classes / Modularité

- **Une classe = un ensemble de variables et de fonctions**
- **class** *identificateur* { *liste\_d'instructions* }
- ***C.f*** pour désigner la fonction *f* de la classe *C*  
Math.random, Integer.parseInt
- ***C.x*** pour désigner la variable *x* de la classe *C*  
Integer.MIN\_VALUE, Integer.MAX\_VALUE, Float.NaN,  
Byte.MIN\_VALUE, Byte.MAX\_VALUE,  
System.out, System.err, System.in
- **le nom de la classe est facultatif pour référencer une variable ou une fonction de la classe courante.**

## Visibilité des noms dans les classes

- par défaut, on accède aux champs des classes du répertoire courant ("**paquetage courant**"),
- on ne peut accéder qu'aux champs **publics** d'une classe figurant dans un autre paquetage (public).
- il y a des restrictions d'accès pour les champs déclarés privés (private).

## Objets

- Les classes sont aussi des **types** de données
- Les objets sont les **valeurs** de type classe.

```
class Date {  
  
    int    j; // jour  
    int    m; // mois  
    int    a; // annee  
}  
  
class BogueVirtuel {  
    public static void main (String[ ] args) {  
  
        Date an0 = new Date();  
        an0.j = 1; an0.m = 1; an0.a= 1970;  
  
        Date an2000 = new Date();  
        an2000.j = 1; an2000.m = 1; an2000.a= 2000;  
  
        System.out.println (an0.j + "/" + an0.m + "/" + an0.a);  
        System.out.println (an2000.j + "/" + an2000.m + "/" + an2000.a);  
    }  
}
```

## Objets – 2

- $C\ o;$  pour déclarer l'objet  $o$  de classe  $C$   
`String s; Date s;`
- $o.f$  pour désigner la méthode  $f$  de l'objet  $o$   
`s.charAt, s.compareTo, System.out.println, System.err.println`
- $o.x$  pour désigner le champ  $x$  de l'objet  $o$   
`d.j, d.m, d.a`

## Objets / Constructeurs

- un **constructeur** est une fonction anonyme (non statique) que l'on peut déclarer dans toute classe et qui retourne un nouvel objet de cette classe.
- un champ (non statique) correspond à une case mémoire **différente** pour tout objet instance de cette classe. On peut l'initialiser à la création de l'objet avec un **constructeur**.

### Exemple

```
class Date {
    int    j, m, a;

    Date (int jour, int mois, int annee) {
        j = jour; m = mois; a = annee;
    }
    ...
    Date an0 = new Date (1, 1, 1970);
    Date an2000 = new Date (1, 1, 2000);
    ...
}
}
```

## Objets / Constructeurs / bis

- dans un constructeur, `this` désigne l'objet en cours de construction. Son utilisation est facultative quand le contexte n'est pas ambigu.

### Exemple revu

```
class Date {
    int    j, m, a;

    Date (int jour, int mois, int annee) {
        this.j = jour; this.m = mois; this.a = annee;
    }

    ...
    Date an0 = new Date (1, 1, 1970);
    Date an2000 = new Date (1, 1, 2000);
    ...
}
```

En fait le constructeur commence implicitement par

```
this = new Date();
```

et finit aussi implicitement par

```
return this;
```

## Objets / Champs statiques

- un champ `static` n'existe qu'en **un seul exemplaire** pour toute la classe.
- Les méthodes ou champs statiques peuvent être indifféremment écrits sous la forme *o.f*, *o.x* ou *C.f*, *C.x* (La deuxième manière étant plus claire).

### Exemple

```
class Date {
    int    j, m, a;
    static int instances = 0;

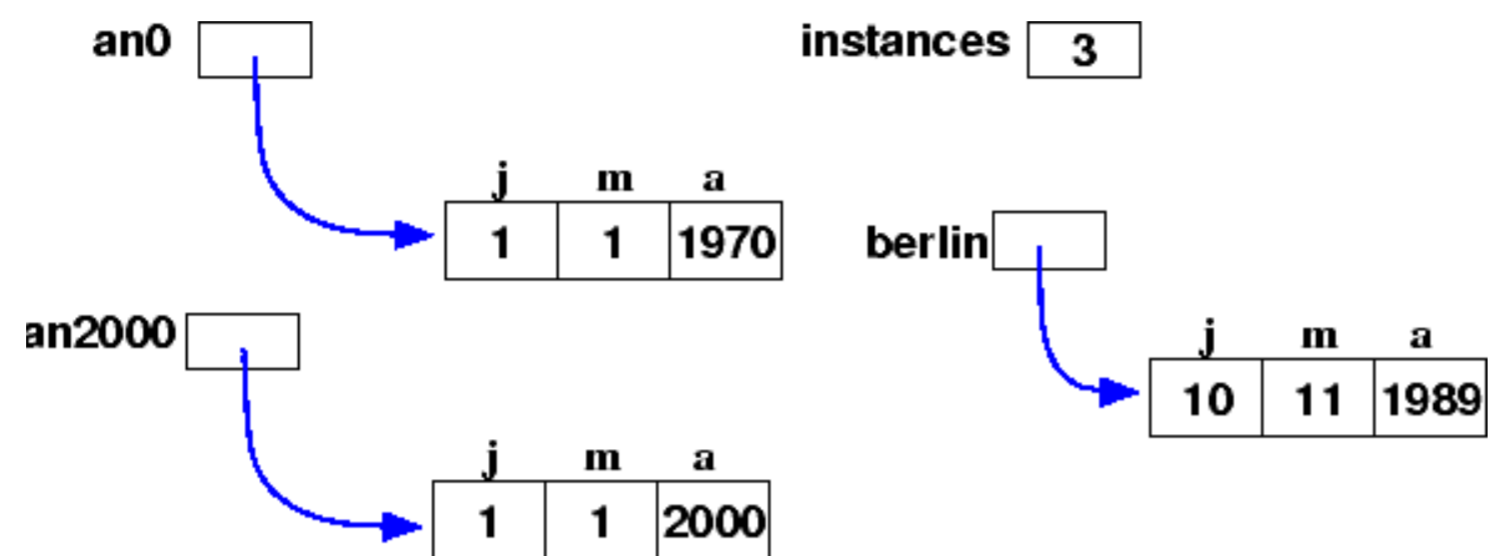
    Date (int jour, int mois, int annee) {
        j = jour; m = mois; a = annee; ++instances;
    }

    static Date an0 = new Date (1, 1, 1970);
    static Date an2000 = new Date (1, 1, 2000);
    static Date berlin = new Date (10, 11, 1989);
}
```

**Exercice 2** Que se passe-t-il si on oublie de mettre les attributs `static` pour les trois dernières dates?

### Objets / Valeur

- la valeur d'un objet est son adresse.



**Exercice 3** Si  $a$  et  $b$  sont deux dates, quel est le sens de l'affectation `a = b` ?

## Objets / Equals / toString

Deux méthodes (non statiques) utiles à définir dans une classe

```
class Date {  
    ...  
    public boolean equals (Date d) {  
        return j == d.j && m == d.m && a == d.a;  
    }  
  
    public String toString () {  
        return j + "/" + m + "/" + a;  
    }  
}
```

Par convention, `toString` est appelé par `+` quand il a le sens de la concaténation des chaînes de caractères. De même pour `System.out.print (x)` qui équivaut à `System.out.print (x + "")` et donc à `System.out.print (x.toString())`.

On peut donc écrire `System.out.println (d)` dans le cas des dates.

## Arguments des méthodes non statiques

Comme pour les constructeurs, l'objet dont la méthode est un champ peut être précisé par le mot-clé `this`.

On peut donc reprendre l'exemple précédent.

```
class Date {  
    ...  
    public boolean equals (Date d) {  
        return this.j == d.j && this.m == d.m && this.a == d.a;  
    }  
  
    public String toString () {  
        return this.j + "/" + this.m + "/" + this.a;  
    }  
}
```

## Grammaire BNF de Java (*Backus Naur Form*)

La grammaire de Java est complètement spécifiée, comme pour  
~tous les langages de programmation.

ForStatement:

```
for ( ForInitopt ; Expressionopt ; ForUpdateopt )  
    Statement
```

ForInit:

```
StatementExpressionList  
LocalVariableDeclaration
```

ForUpdate:

```
StatementExpressionList
```

StatementExpressionList:

```
StatementExpression  
StatementExpressionList , StatementExpression
```

StatementExpression:

Assignment

PreIncrementExpression

PreDecrementExpression

PostIncrementExpression

PostDecrementExpression

MethodInvocation

ClassInstanceCreationExpression

**Tri**

et

**Recherche en table**

## Recherche d'un minimum dans un tableau

```
static int minimum (int[ ] a) {  
    r = Integer.MAX_VALUE;  
    for (i=0; i < a.length; ++i)  
        if ( a[i] < r )  
            r = a[i];  
    return r;  
}
```

Ca marche pour le cas vide. (Il faut toujours y penser!).

- $n$  comparaisons (si  $n$  est la longueur du tableau)
- nombre total d'opérations est  $O(n)$   
(algorithme linéaire en temps),  
Il a une **complexité** linéaire.

## Tri sélection

**Donné:** un tableau  $a$  de  $n$  éléments.

**Résultat:** le même tableau trié en ordre croissant, c'est à dire  $a_i \leq a_j$  pour  $0 \leq i < j < n$ .

**Algorithme:**

- On cherche l'emplacement du minimum de  $a$ ,
- On échange l'élément de tête de  $a$  avec le minimum
- On recommence sur le tableau moins le premier élément, ... tant que le tableau à traiter n'est pas vide.

## Echange de deux éléments

- pour échanger les valeurs de deux variables, on effectue une permutation circulaire avec une troisième variable (auxiliaire).

```
int x = 1, y = 2;  
  
int aux = x;  
x = y;  
y = aux;
```

Peut-on faire mieux?

## Tri sélection – le programme

```
static void triSelection (int[ ] a) {
    int  i_min, t;

    for (int i = 0; i < a.length - 1; ++i) {
        i_min = i;
        for (int j = i+1; j < a.length; ++j)
            if (a[j] < a[i_min])
                i_min = j;
        t = a[i_min]; a[i_min] = a[i]; a[i] = t;
    }
}
```

- $n^2$  comparaisons (si  $n$  est la longueur du tableau)
- nombre total d'opérations est  $O(n^2)$  (algorithme quadratique en temps), Il a une **complexité** quadratique.

## Tri par insertions

Comme pour trier un paquet de cartes

```
static void triInsertion (int[ ] a) {
    int j, v;

    for (int i = 1; i < a.length; ++i) {
        v = a[i]; j = i;
        while (j > 0 && a[j-1] > v) {
            a[j] = a[j-1];
            --j;
        }
        a[j] = v;
    }
}
```

- $I(a)$  comparaisons (le nombre d'inversions dans  $a$ )
- nombre total d'opérations est  $O(n^2)$  dans le cas pire (algorithme quadratique en temps),  
Il a une **complexité** quadratique.
- bon quand  $a$  est presque trié.

## Le tri Shell [D. L. Shell 1959]

```
static void triShell (int[ ] a) {
    int h = 1; do h = 3*h + 1; while ( h <= a.length );
    do {
        h = h / 3;
        for (int i = h; i < a.length; ++i)
            if (a[i] < a[i-h]) {
                int v = a[i], j = i;
                do {
                    a[j] = a[j-h];
                    j = j - h;
                } while (j >= h && a[j-h] > v);
                a[j] = v;
            }
    } while ( h > 1);
}
```

**Pas plus que  $O(n^{3/2})$  comparaisons !!**  
**Bon pour de gros fichiers (celui du noyau Maple)**

## Recherche en table

Une table est un ensemble de paires  $(k, d)$  où  $k$  est une clé et  $d$  une donnée. Par exemple,  $k$  est un nom de personne,  $d$  est son numéro de téléphone (ou son adresse e-mail).

On cherche une clé dans une table pour obtenir la donnée correspondante.

### Recherche séquentielle

```
static int recherche (String x, String[ ] nom, int[ ] tel) {  
    for (int i = 0; i < nom.length; ++i)  
        if (x.equals(nom[i]))  
            return tel[i];  
    return -1;  
}
```

## Recherche en table – bis

nom	tel
Robert	05 56 84 60 88
Jean-Jacques	01 39 63 56 89
Philippe	01 69 33 34 89
Sam	01 69 33 46 15
Catherine	01 69 33 34 67
Guillaume	03 83 59 30 21

## Recherche dichotomique en table

On suppose la table ordonnée en ordre croissant.

```
static int rechercheDichotomique (String x, String[] nom, int[] tel) {
    int i, cmp, g = 0, d = nom.length-1;
    do {
        i = (g + d) / 2;
        cmp = x.compareTo(nom[i]);
        if (cmp == 0)
            return tel[i];
        if (cmp < 0)
            d = i - 1;
        else
            g = i + 1;
    } while (g <= d);
    return -1;
}
```

Complexité  $O(\log n)$ .

## Recherche par interpolation

Si distribution uniforme, interpolation linéaire de la clé recherchée. Complexité  $O(\log(\log n))$ .

## Hachage – Hashing

- **Idée:** on range l'élément de clé  $x$  en position  $h(x) \in [0, M - 1]$ , où  $h$  est une fonction "aléatoire".

- **Prérequis :** avoir une "bonne" fonction  $h$ .

**Solution,** si  $x = x_0x_1 \dots x_{n-1}$  est une chaîne de caractères :

$$h(x) = (x_0 * B^{n-1} + x_1 * B^{n-2} + \dots + x_{n-1}) \bmod M$$

```
static int h (String x) {  
    int    r = 0;  
    for (int i = 0; i < x.length(); ++i)  
        r = ((r * B) + x.charAt(i)) % M;  
    return r;  
}
```

**Par exemple choisir  $M$  premier, et  $B = 128$  ou  $B = 256$ .**

## Hachage et collisions

Si  $h(i) = h(j)$  "par hasard", que faire?

Plusieurs solutions :

- Hachage avec listes de collision
- Hachage avec adressage ouvert
- Hachage à 2 niveaux avec adressage ouvert
- Hachage avec chaînage
- Hachage multiple
- Hachage pour cache

## Listes de collisions (listes d'association)

```
final static int M = 63, B = 128;

static Entree[] nom = new Entree[M];

static int rechercher (String x) {
    for (Entree e = nom[h(x)]; e != null; e = e.suivant)
        if (x.equals(e.cle))
            return e.tel;
    return -1;
}

static void inserer (String x, int t) {
    int i = h(x);
    nom[i] = ajouter (x, t, nom[i]);
}

static Entree ajouter (String x, int t, Entree e) {
    if (e == null)
        return new Entree (x, t, null);
    else {
        e.suivant = ajouter (x, t, e.suivant);
        return e;
    }
}
```

## Hachage par Adressage ouvert

On range directement les entrées dans le tableau, et en cas de collision on fait une bête recherche linéaire. Il faut pouvoir borner le nombre maximum d'entrées. On utilise une valeur spéciale pour repérer les cases libres.

```
static int rechercher (String x) {
    for (int i = h(x); !nom[i].equals (""); i = (i+1) % M)
        if (x.equals(nom[i]))
            return tel[i];
    return -1;
}

static void inserer (String x, int t) {
    int i = h(x);
    while (!nom[i].equals (""))
        i = (i+1) % M;
    nom[i] = x; tel[i] = t;
}
```

## Adressage ouvert – bis

La complexité ne dépend que du taux d'occupation  $\alpha = n/m$ , et vaut:

$$\frac{1}{2} \left(1 + \frac{1}{1-\alpha}\right)^2 \quad \text{pour une insertion ou recherche avec échec,}$$

$$\frac{1}{2} \left(1 + \frac{1}{1-\alpha}\right) \quad \text{pour une recherche avec succès.}$$

Pour  $\alpha = 66\%$ , on fait 5 et 2.

Pour  $\alpha = 90\%$ , on fait 50 et 5.

Pour accélérer on peut faire un double hachage, en augmentant  $i$  de  $u = h_2(x)$ , où  $h_2$  est une seconde fonction de hachage, au lieu du pas de 1.

On fait alors  $\frac{1}{1-\alpha}$  et  $\frac{1}{\alpha} \ln \left(\frac{1}{1-\alpha}\right)$ .

Pour  $\alpha = 80\%$ , 5 et 1,3;

Pour  $\alpha = 99\%$ , 100 et 5.

## Hachage multiple

- On peut hacher une grande table (dictionnaire par exemple) de  $d$  mots par  $k$  fonctions  $h_i$  indépendantes,  $1 \leq i \leq k$ . Et on prend un grand tableau  $T$  de  $n$  bits où  $T[j] = 1$  si  $h_i(v) = j$  pour  $v$  dans le dictionnaire et  $1 \leq i \leq k$ .
- La probabilité pour qu'aucun mot d'un dictionnaire de taille  $d$  ne positionne un  $T[j]$  donné est

$$p = e^{-\alpha} \quad \text{où } \alpha = dk/n.$$

La probabilité pour qu'un  $v$  arbitraire soit dans le dictionnaire est  $(1 - p)^k$ .

Ce qui se minimise pour  $d = 25000$ ,  $n = 400000$ ,  $k = 11$  en 0.000458711.

- C'est la méthode utilisée par la commande `spell` de Unix.

**Exercice 4** Calculer la complexité moyenne de la recherche avec hachage simple dans le cas d'une recherche avec succès et avec échec.

**Exercice 5** Si on insère les nouvelles entrées en tête de la liste de collisions plutôt qu'à la fin, cela change-t-il le coût moyen?

**Exercice 6** Calculer la complexité du tri par insertions.

**Exercice 7** (plus difficile) Montrer les formules de complexité de la recherche avec succès et échec du hachage avec adressage ouvert.

## Renseignements sur Java

Aller voir les pages Web en:

```
http://www.enseignement.polytechnique.fr/local/documentation/  
.../documentation/java/jdk1.1.7/docs/  
.../documentation/java/jdk1.1.7/docs/api/packages.html
```