Introduction à l'Informatique (INF 311)

F. Morain

Amphi 6: tables

6 juin

I. Pour se réveiller en douceur: MacLib.

II. Organiser l'information.

III. Recherche séquentielle.

IV. Recherche dichomotique.

0. Où en sommes-nous?

- Amphi 1: introduction.
- Amphi 2: programmer en Java.
- Amphi 3: fonctions/fonctions récursives.
- Amphi 4: tableaux/String.
- Amphi 5: classes.
- Amphi 6: tables.
- Amphi 7: algorithmes et complexité.
- Amphi 8: Internet.
- Amphi 9: listes.
- Amphi 10: système/sécurité.

I. Pour se réveiller en douceur: MacLib

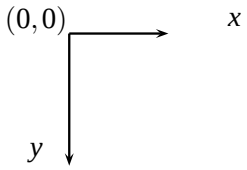
Un exemple de longévité pour un logiciel !

Librairie graphique mise au point par Philippe Chassignet.

- Jusqu'en 1992: **TGiX** (Unix vers Macintosh).
- 1992–94: **ThinkPascal** sur Mac; PC-Windows, **TurboPascal**.
- 1996–98: **ThinkC** et **TurboC**, X11; **DelphiPascal**, **Borland C** (nouveau Windows); **CodeWarrior Pascal et C**.
- 1998: Java $\Rightarrow$ une seule version pour toutes les plateformes ! Interfacée avec l'AWT (*Abstract Windowing Toolkit*), usage souple de la boucle d'événement.

Principes: à la Macintosh

- **`QuickDraw`**: coordonnées entières (**`int`**);



- point courant: on imagine qu'on a en main un crayon qui repose en un point;
- primitives simples: **`moveTo`**, **`lineTo`**;
- quelques objets: points, rectangles;
- couleurs.
- etc. (cf. poly).

Un exemple simple

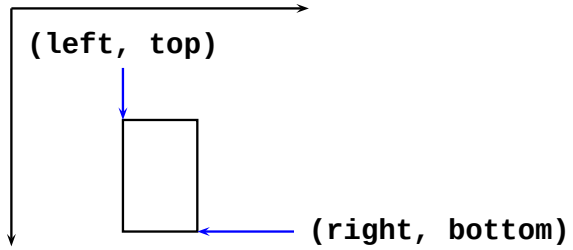
```
public class Dessins{

  public static void Dessin1(){
    MacLib.moveTo(0, 0);
    MacLib.lineTo(100, 100);
  }

  public static void main(String[] args){
    // il faut initialiser MacLib
    MacLib.initQuickDraw();
    Dessin1();
  }
}
```


Rectangles

```
public class Rect{
  short left, top, right, bottom;
}
```



Quelques fonctions: `setRect(r, g, h, d, b)`, `frameRect(r)`, `paintRect(r)`, etc. (cf. poly).

Exemple

```
public class Dessin{
  public static void main(String[] args){
    Rect r = new Rect();

    MacLib.initQuickDraw();
    MacLib.setRect(r, 20, 20, 100, 100);
    MacLib.paintRect(r);

    MacLib.foreColor(MacLib.yellowColor);
    MacLib.setRect(r, 40, 40, 80, 80);
    MacLib.paintRect(r);
  }
}
```


II. Organiser l'information

Pb: soit E un ensemble de données. Comment le stocker ? Et de la façon la plus efficace possible ? Comment rechercher les informations présentes, insérer/supprimer facilement, etc.?

Démarche: on modélise l'information, puis on choisit dans le grand catalogue:

- donnée composite → objet ;
- nombreuses données de même type sans relation entre elles → table ;
- lien hiérarchique entre les données → arbre, tas ;
- liens intriqués (réseau routier, etc.) → graphes.

L'exemple conducteur

Fil rouge: un annuaire permet de stocker des couples (abonné, numéro de téléphone).

Fonctionnalités demandées:

- trouver le numéro de téléphone d'un abonné donné;
- trouver un abonné de numéro donné.

Solution la plus simple à programmer: on stocke tout dans un tableau.

Modèle de données: les numéros de téléphone sont uniques, mais pas les noms.

Classes de base

```
public class Abonne{
  String nom, tel;

  public Abonne(String n, String t){
    this.nom = n;
    this.tel = t;
  }
}
public class Annuaire{
  String operateur;
  Abonne[] a;

  public Annuaire(String o, int nab){
    this.operateur = o;
    this.a = new Abonne[nab];
  }
```



```
  public static void afficher(Annuaire A){
   System.out.print("Annuaire de l'opérateur ");
   System.out.println(A.operateur);
   for(int i = 0; i < A.a.length; i++)
    System.out.println(A.a[i].nom+" "+A.a[i].tel);
  }
  // exemple jouet pour le débogage
  public static Annuaire initialiser(){
    Annuaire A = new Annuaire("ex", 6);
    A.a[0] = new Abonne("paul", "0607082811");
    A.a[1] = new Abonne("sabrina", "0607084501");
    A.a[2] = new Abonne("laure", "0607082701");
    A.a[3] = new Abonne("sandrine", "0607082702");
    A.a[4] = new Abonne("paul", "0607082805");
    A.a[5] = new Abonne("yves", "0607082806");
    return A;
  }
  public static void main(String[] args){
    Annuaire A = initialiser();
    afficher(A);
  }
}
```


Lecture des données depuis un fichier

Quel format? `nom tel` comme dans le fichier correspondant au fichier jouet:

```
RFS
paul 0607082811
sabrina 0607084501
laure 0607082701
sandrine 0607082702
paul 0607082805
yves 0607082806
```

Rem. Chaque opérateur a son chiffre magique après le **`06`**. Il doit être le même pour tous les abonnés. Cela lui permet d'avoir 10^7 clients. . .

Extension possible: (en exercice) pour résister aux noms avec un espace, un caractère spécial, etc., on peut choisir **`';'`** comme séparateur (cf. format **`.csv`** d'Excel ultra portable).

```
  public static Annuaire deFichier(String nomfic){
    String[] lignes = TC.lignesDeFichier(nomfic);
    String operateur = lignes[0];
    int nab = lignes.length-1;
    Annuaire A = new Annuaire(operateur, nab);
    for(int i = 1; i < lignes.length; i++){
      // lignes[i] = "nom tel"
      String[] mots = TC.motsDeChaine(lignes[i]);
      A.a[i-1] = new Abonne(mots[0], mots[1]);
    }
    return A;
  }
  public static void main(String[] args){
    Annuaire A = deFichier(args[0]);

    afficher(A);
  }
```


Et si on parlait de robustesse?

Problème: et si le fichier de données est corrompu? On peut s'en apercevoir à la lecture, en détectant que la ligne contient bien deux champs séparés par un blanc, le second avec 10 caractères formés de chiffres, commençant par **"06"**, suivi du chiffre magique.

Rem. On pourrait aller plus loin en vérifiant que tous les caractères du mot de tête sont valides.

Principe: la fonction **estEntreeCorrecte** retourne un booléen indiquant si les entrées sont bonnes ou pas.


```
public static Annuaire deFichier(String nomfic){
  String[] lignes = TC.lignesDeFichier(nomfic);
  String operateur = lignes[0];
  int nab = lignes.length-1;
  Annuaire A = new Annuaire(operateur, nba);
  for(int i = 1; i < lignes.length; i++){
    // lignes[i] = "nom tel"
    String[] mots = TC.motsDeChaine(lignes[i]);

    if(!estEntreeCorrecte(mots))
      return null;
    A.a[i-1] = new Abonne(mots[0], mots[1]);
  }
  return A;
}
public static void Erreur(String s){
  System.err.println("Erreur: "+s);
}
```



```
public static boolean
          estEntreeCorrecte(String[] mots){
  if(mots.length != 2){
    Erreur("Je n'ai pas deux champs!");
    return false;
  }
  if(mots[1].length() != 10){
    Erreur("Mauvaise longueur : "+mots[1]);
    return false;
  }
  if((mots[1].charAt(0) != '0')
     || (mots[1].charAt(1) != '6')){
    Erreur("Mauvais début : "+mots[1]);
    return false;
  }
  for(int i = 2; i < mots[1].length(); i++)
    if(! Character.isDigit(mots[1].charAt(i))){
      Erreur("Ce n'est pas un chiffre :"
           +mots[1].charAt(i));
      return false;
    }
  return true;
}
```

Simulons de vraies données

Chaque opérateur gère quelques millions d'abonnés. On peut générer des données aléatoires facilement, qu'on mettra dans un fichier.
Pour avoir des numéros uniques, on les numérote de 1 en 1.

```
public class Generer{
  public static void main(String[] args){
    // on affiche le nom de l'opérateur
    System.out.println(args[0]);
    int magique = Integer.parseInt(args[1]);
    for(int i = 0; i < Integer.parseInt(args[2]); i++){
      System.out.print(nomAleatoire());
      System.out.println(" "+tel(magique, i));
    }
  }
}
```

```
unix% java Generer rouge 4 1000 > rouge.in
unix% java Generer jaune 5 1000000 > jaune.in
```



```
public static String nomAleatoire(){
  String alphabet = "abcdefghijklmnopqrstuvwxyz";
  // calcul de la longueur
  int nl = 4 + (int)(Math.random() * 5);
  // fabrication de la chaîne
  String n = "";
  for(int i = 0; i < nl; i++){
    // on tire un caractère au hasard
    int j=(int)(Math.random()*alphabet.length());
    n += alphabet.charAt(j);
  }
  return n;
}
```

Amélioration possible: on fabrique des noms au hasard avec une distribution plus proche de la réalité. Par exemple, pour la promo X2005, on obtient les proportions suivantes (pour les premières lettres):

a	b	c	d	e	f	g	h	i	j	k	l	m
19	54	48	52	9	10	32	16	3	12	11	51	49

n	o	p	q	r	s	t	u	v	w	x	y	z
11	2	31	2	19	20	18	1	9	11	1	4	5


```
  public static String tel(int magique, int i){
    // convertit i en String
    String t = String.valueOf(i);

    // on rajoute des zéros si besoin
    while(t.length() != 7)
      t = "0"+t;
    return "06"+magique+t;
  }
```

Testons:

```
unix% java -Xmx256m Annuaire jaune.in
```

(on a besoin d'étendre la taille de la mémoire utilisée par Java).

III. Recherche séquentielle

Principe: pour trouver le numéro de téléphone d'un abonné de nom donné, on parcourt tout l'annuaire pour trouver les abonnés ayant ce nom.

Entrée de la fonction: un nom (de type **`String`**).

La fonction la plus évidente: affichage à l'écran seulement

```
public static void rechercher(Annuaire A,
                              String x){
  for(int i = 0; i < A.a.length; i++)
    if(x.equals(A.a[i].nom))
      System.out.println(A.a[i].tel);
}
public static void main(String[] args){
  Annuaire A = deFichier(args[0]);
  rechercher(A, "paul");
}
```

Problèmes:

- peu convivial;
- mélange recherche et affichage;
- on ne peut rien faire avec les données.

Comment améliorer l'ergonomie?

```
public static void recherches(Annuaire A){
  while(true){
    System.out.print("Entrer un nom ");
    System.out.print("('Q' pour quitter)=");
    String nom = TC.lireLigne();
    if(nom.equals("Q"))
      break;
    rechercher(A, nom));
  }
}
public static void main(String[] args){
  Annuaire A = initialiser();
  recherches(A);
}
```


À propos du **break**

Syntaxe: pour sortir d'une boucle **while** ou **for**:

```
while(Condition){
  ...
  break;
}
```

En fait: on sort de la boucle la plus interne

```
int[][] t = {{1, 2}, {2, 3}, {3, 4}};
for(i = 0; i < t.length; i++){
  for(j = 0; j < t[i].length; j++)
    if(t[i][j] == 1)
      break;
  System.out.print("Après j="+j+" ");
}
System.out.println("Fini");
```

L'affichage est:

```
Après j=0 Après j=2 Après j=2
Fini
```


Amélioration possible: on met dans un tableau les numéros de téléphone trouvés.

Encore mieux: on stocke les indices des numéros, ce qui nous permet d'accéder à d'autres données si nécessaire.

```
// retourne le nombre de numéros trouvés
public static int rechercher2(int[] t,
                              Annuaire A,
                              String x){
  int nt = 0;

  for(int i = 0; i < A.a.length; i++)
    if(x.equals(A.a[i].nom))
      t[nt++] = i;
  return nt;
}
```



```
public static void recherches(Annuaire A){
  int[] t = new int[A.a.length];
  int nt;
  while(true){
    System.out.print("Entrer un nom ");
    System.out.print("('Q' pour quitter)=");
    String nom = TC.lireLigne();
    if(nom.equals("Q"))
      break;
    nt = rechercher2(t, A, nom);
    afficher(A, nom, t, nt);
  }
}
public static void afficher(Annuaire A,
                            String nom,
                            int[] t, int nt){
  System.out.print("Voici les "+nt+" réponses de"
  System.out.println(" "+A.operateur+" pour "+nom
  for(int i = 0; i < nt; i++)
    System.out.println(A.a[t[i]].tel);
}
```

Variante avec plusieurs opérateurs

```
public static void recherches(Annuaire[] A){
  while(true){
    System.out.print("Entrer un nom ");
    System.out.print("('Q' pour quitter)=");
    String nom = TC.lireLigne();
    if(nom.equals("Q"))
      break;
    for(int i = 0; i < A.length; i++){
      int[] t = new int[A[i].a.length];
      int nt = rechercher2(t, A[i], nom);
      afficher(A[i], nom, t, nt);
    }
  }}
public static void main(String[] args){
  int nbA = args.length;
  Annuaire[] A = new Annuaire[nbA];
  for(int i = 0; i < nbA; i++)
    A[i] = deFichier(args[i]);
  recherches(A);   }
```

```
unix% java -Xmx256m Annuaire jaune.in rouge.in
```


Application: croisement des annuaires

Problème: déterminer qui a un abonnement chez plusieurs opérateurs.
Principe: pour chaque nom de **A2**, on regarde si on le trouve également dans **A1**.

```
public static void croiser(Annuaire A1,
                           Annuaire A2){
  int[] t = new int[A1.a.length];
  int nt;

  for(int i = 0; i < A2.a.length; i++){
    nt = rechercher2(t, A1, A2.a[i].nom);
    if(nt > 0){
      System.out.print(A2.a[i].nom);
      System.out.print(" est dans les deux");
      System.out.println(" annuaires");
    }
  }
}
```


Performances en pratique

Combien de temps prend une requête en fonction de $n =$ taille de l'annuaire?

Il y a n comparaisons de chaînes à effectuer dans tous les cas.

Dans le cas du croisement, le coût est $|A_1| \times |A_2|$.
Peut-on faire mieux?

Oui, si le tableau est trié. Si un mot commence par **a**, inutile de continuer les comparaisons au-delà des noms commençant par **a**.

À propos du tri

On en verra quelques variantes la semaine prochaine. Cela peut être fait rapidement.
En Unix, on peut écrire :

```
unix% sort -k1 jaune.in > jaune.trie
```

qui trie par rapport à la première colonne (les noms des abonnés).
Si on veut trier par rapport aux numéros de téléphone, on écrit:

```
unix% sort -k2 jaune.in > jaune.tel
```

Pour plus d'infos (en plus de l'amphi 7) :

```
unix% man sort
unix% info sort
```

Modification immédiate de la recherche

Principe: on compare la cible **x** avec les mots de l'annuaire, et on s'arrête dès qu'on a trouvé un mot placé après **x** dans l'ordre lexicographique. On gagne beaucoup quand **x** commence par **a**.

```
// on suppose A trié
public static int rechercher3(int[] t,
                              Annuaire A,
                              String x){
  int nt = 0;
  for(int i = 0; i < A.a.length; i++){
    int cmp = x.compareTo(A.a[i].nom);
    if(cmp == 0)
      t[nt++] = i;
    else if(cmp < 0)
      // x est avant A.a[i].nom
      // dans l'ordre lexicographique
      break;
  }
  return nt;
}
```


Modification plus intelligente: index

On effectue un prétraitement de l'information.

But: comme dans un carnet d'adresses, accéder directement à la zone des **a** à l'aide d'un onglet.

Comment ? stocker dans un tableau auxiliaire l'indice du début de la zone des **a**.

Principe: l'index sera tel que **`ind[0]`** sera l'indice du premier nom dans **`A.a`** commençant par **a**, . . . , **`ind[25]`** l'indice du premier nom commençant par **z**.

Autrement dit, l'intervalle $[\text{ind}[0], \text{ind}[1][$ est l'ensemble (éventuellement vide) des indices i tel que **`A.a[i]`** commence par la lettre **a**.

Enrichissez-vous!

On va enrichir la structure de données.

```
public class Annuaire{
  String operateur;
  Abonne[] a;
  int[] index;

  public Annuaire(String o, int nab){
    this.operateur = o;
    this.a = new Abonne[nab];
    this.index = new int[27]; // cf. plus loin
  }
}
```

Fonction auxiliaire pratique: associe à une lettre minuscule son code entre 0 et 25.

```
public static int indiceChar(char c){
  String alphabet = "abcdefghijklmnopqrstuvwxyz";
  for(int i = 0; i < alphabet.length(); i++)
    if(alphabet.charAt(i) == c)
      return i;
  return -1;
}
```


La recherche proprement dite

```
public static int rechercher4(int[] t,
                              Annuaire A,
                              String x){
  int ic = indiceChar(x.charAt(0));
  int imin = A.index[ic];
  int imax = A.index[ic+1];
  int nt = 0;
  for(int i = imin; i < imax; i++){
    int cmp = x.compareTo(A.a[i].nom);
    if(cmp == 0)
      t[nt++] = i;
    else if(cmp < 0)
      // x est avant A.a[i].nom
      // dans l'ordre lexicographique
      break;
  }
  return nt;
}
```

Construction de l'index

Principe: on parcourt le tableau déjà construit et on repère les changements de première lettre.

Exemple avec `jouet.trie`:

```
RFS
laure 0607082701
paul 0607082805
paul 0607082811
sabrina 0607084501
sandrine 0607082702
yves 0607082806
```

a	b	c	d	e	f	g	h	i	j	k	l	m
?	?	?	?	?	?	?	?	?	?	?	0	?

n	o	p	q	r	s	t	u	v	w	x	y	z
?	?	1	?	?	3	?	?	?	?	?	5	?

Attention aux cas pathologiques:

- s'il n'y a pas de mot commençant par une lettre donnée;
- si aucun mot ne commence par **a** ou **z**.

Astuce: pour construire des intervalles vides: on met dans **ind[i]** la valeur **n = A.a.length**.

Dans l'exemple:
ind[0]=6, **ind[1]=6** et donc l'intervalle $[\text{ind}[0], \text{ind}[1][= [6, 6[= \emptyset$;
$\Rightarrow$ il n'y a pas de mot commençant par **a**.

De même, $[\text{ind}[10], \text{ind}[11][= [6, 0[= \emptyset$, $[\text{ind}[24], \text{ind}[25][= [5, 6[= \{5\}$.

Reste le cas de **z**: $[\text{ind}[25], \text{ind}[26][= [6, 6[= \emptyset$. Ça explique le 27!!!


```
public static void creerIndex(Annuaire A){
  int ia, ic, n = A.a.length;
  char lettre;

  for(int i = 0; i < 27; i++)
    A.index[i] = n; // astuce
  ic = 0; // indice courant dans A.a[]
  while(true){
    // début d'une nouvelle lettre
    // A.a[ic] commence par 'lettre'
    lettre = A.a[ic].nom.charAt(0);
    ia = indiceChar(lettre);
    A.index[ia] = ic;
    do{
      ic++;
    } while((ic < n) // attention aux conditions
        && (A.a[ic].nom.charAt(0) == lettre));
    if(ic >= n)
      // on est au bout de l'annuaire
      break;
  }
}
```


Quel est le gain?

Si les données sont réparties uniformément, alors on divise le temps par 26.

On peut améliorer en fabriquant un index adapté aux données; par exemple **aa**, etc.

IV. Recherche dichotomique

Cas particulier important: les données stockées sont uniques $\Rightarrow$ au plus une solution à la recherche. C'est le cas pour les numéros de téléphone.

Principe: on s'arrête tout de suite si on a trouvé. En moyenne, on va faire $n/2$ accès.

Peut-on faire mieux? oui si les données sont triées, ce qui est le cas des fichiers créés par **`Generer`**.

Principe: comme la recherche dans un dictionnaire. On ouvre à une page au milieu, puis on regarde si on doit chercher dans la moitié droite ou la moitié gauche.


```
public static int rechercherDicho(Annuaire A,
                                  String tel){
  int g = 0, d = A.a.length-1, m;

  // on recherche dans [g, d]
  while(g <= d){
    m = (g+d)/2;
    int cmp = tel.compareTo(A.a[m].tel);
    if(cmp == 0)
      return m;
    else if(cmp < 0)
      // tel est avant A.a[i].tel
      d = m-1;
    else // tel est après A.a[i].tel
      g = m+1;
  }
  // on n'a rien trouvé
  return -1;
}
```


Version récursive

```
public static int dichoRec(Annuaire A,
                           String tel,
                           int g, int d){
  if(g > d)
    return -1;
  int m = (g+d)/2;
  int cmp = tel.compareTo(A.a[m].tel);
  if(cmp == 0)
    return m;
  else if(cmp < 0)
      // tel est avant A.a[i].tel
    return dichoRec(A, tel, g, m-1);
  else // tel est après A.a[i].tel
    return dichoRec(A, tel, m+1, d);
}
```


Combien l'algorithme fait-il de comparaisons ?

À chaque appel de **`dichoRec`**, on effectue une comparaison.
On entre dans la fonction avec l'intervalle $[g,d]$. Le cas intéressant est pour $g \leqslant d$.
On appelle récursivement la fonction sur l'intervalle $[g',d']$ avec $[g',d'] = [g,m-1]$ ou $[m+1,d]$.
La longueur de l'intervalle $[g,d]$ est $d-g+1$.

Lemme. $d'-g'+1 = \lfloor \frac{d-g}{2} \rfloor$.

$\Rightarrow$ la longueur de l'intervalle est divisée par 2 à chaque étape.
$\Rightarrow$ on fera au plus $\log n$ appels, donc au plus $\log n$ comparaisons.

Ex. Pour $n = 10^6$, on passe de $500,000$ comparaisons à 19.

Où en sommes-nous ?

On a vu comment chercher dans une table de taille n, en cn opérations, ou $\log n$ si on a un ordre sur les données.

Si la table est petite: recherche séquentielle suffit.

Si la table est grande:

- s'il existe un ordre sur les données: tri $+$ recherche dichotomique.
- s'il n'y a pas d'ordre ou la table est trop grande: hachage, avec insertion et recherche en $O(1)$; cf. INF-421.

Derniers mots

Prochains rendez-vous:

Groupes	TD 6 (**demain!**)
1–6	13h30–15h30
7–12	10h15–12h15

Prochain amphi: **lundi 12 juin à 10h30 en amphi Arago**.

Tutorat: 8, 15 juin.

Pale machine: 19 juin (suite à l'amphi 8); présence indispensable. . . .

Délégués: réunion de département jeudi 8 juin 16h.