NFP 136 – VARI 2

Algorithmique et Structures de Données

Prolongement de la partie « Algorithmique et Programmation » du cours NFP 135 (VARI 1)

Informatique
= traitement automatique
de l'information

Ingrédients nécessaires :

- Un *cadre formel* pour analyser et décrire ces traitements (*algorithmique*)
- Un *appareil* permettant d'effectuer ces traitements = un ordinateur (ayant sa propre architecture, et un système d'exploitation pour gérer ses interactions)
- Un *langage* pour communiquer avec l'appareil

L'Informatique a de multiples facettes

- C'est à la fois une science : fondements théoriques de la calculabilité et de l'algorithmique, principes de fonctionnement et arithmétique des ordinateurs, etc.

- C'est également, par nature, un domaine de savoirs technologiques : mise en œuvre par la programmation (syntaxe et bonnes pratiques de l'utilisation des langages de programmation), et applications directes dans le monde réel (vidéo, son, interfaces hommes-machines, jeux vidéo, etc.).

- Dans le prolongement de NFP 135, cette UE se focalise sur la présentation des structures de données « usuelles » et des algorithmes associés, mais également sur leur implémentation et la prise en main d'un langage de haut niveau, ainsi que sur des connaissances de base liées aux systèmes d'exploitation.

Objectifs de cette partie du cours :

- Pouvoir manipuler efficacement des données plus nombreuses (et plus complexes).

- Ce qui nécessite de les stocker dans des structures de données « intelligentes », de façon à ce que les opérations (algorithmes) avec lesquelles on les exploite soient le plus efficace possible.

- Les informaticiens, au cours du temps, ont défini des Structures de Données *usuelles* (tableaux, listes, arbres, graphes, etc.) et des Algorithmes *usuels* (tri, manipulation de données structurées, etc.).

- Avec un soucis d'abstraction et de modularité : on cherche à dissocier une structure de données de la façon dont elle est implémentée (notion de *Type Abstrait*).

Contenu :

- Principe des systèmes d'exploitation (permettant d'exécuter des programmes sur un ordinateur).

- Structures de données élémentaires (tableaux, tables de hachage, listes, piles, files) et opérations élémentaires associées (création, parcours, tri).

- Mesure d'efficacité de ces algorithmes (et des autres) : notions de complexité.

- Des structures plus complexes dédiées : graphes (et automates), arbres (binaire ou non) et tas.

Contenu :

⇒ Langage d'implémentation : JAVA

Car :

- Langage de haut niveau et portable,
- Facilite l'abstraction et la programmation modulaire par la structuration en classes d'objets,
- Dispose d'un mécanisme de « garbage collector » (ramasse-miettes) rendant inutile la désallocation « à la main » de l'espace mémoire inutilisé.

DÉFINITION D'UN ALGORITHME

- **Procédure de calcul bien définie qui prend en entrée un ensemble de valeurs et produit en sortie un ensemble de valeurs.** (Cormen, Leiserson, Rivert)

- **Ensemble d'opérations de calcul élémentaires, organisé selon des règles précises dans le but de résoudre un problème donné. Pour chaque donnée du problème, il retourne une réponse après un nombre fini d'opérations.** (Beauquier, Berstel, Chrétienne)

- **<u>Plus simplement</u> : suite de traitements à appliquer à un ensemble d'informations, appelées données, en général dans le but de résoudre un problème précis.**

- Phase 1 – **FORMALISATION DU PROBLÈME**

  Spécification

- Phase 2 – **CONCEPTION ET PREUVE D'UN ALGORITHME POUR CE PROBLÈME**

  Algorithmique

- Phase 3 – **MESURE DE SON EFFICACITÉ**

  Complexité

- Phase 4 – **MISE EN OEUVRE (ET TESTS)**

  Programmation (Java)

UN EXEMPLE D'ALGORITHME : COMMENT RESOUDRE LE « PROBLEME DU BOULANGER » ?

- **Comment un boulanger doit-il rendre la monnaie à un client pour utiliser le moins de billets/pièces possible ?**
- **Algorithme *glouton* : rendre la monnaie à l'aide de la pièce/du billet de plus forte valeur, puis passer au suivant si besoin, etc. Par exemple, si somme = 98 euros, alors le glouton renvoie 50+2*20+5+2+1**
- **Dans notre système (500, 200, 100, 50, 20, 10, 5, 2, 1, 0.50, 0.20, 0.10, 0.05, 0.02, 0.01), la stratégie gloutonne est *TOUJOURS* la meilleure !**
- **Mais si 5, 4, 1 et non 5, 2, 1, alors la stratégie gloutonne n'est *PAS* la meilleure pour 98 !**

Autre exemple : tri d'éléments via un algorithme de tri par sélection

PROBLEME DU TRI :

Entrée : un ensemble de n éléments qui peuvent être ordonnés (par exemple, des nombres, des lettres, des mots, etc.).

Sortie : ces n éléments, triés selon un certain ordre (par exemple, des nombres triés par ordre croissant, ou décroissant) .

Savoir trier est essentiel !

- SOLDES
- BONS PLANS
- VENTES FLASH
- OFFRES ADHÉRENTS
- OFFRES REMBOURSEMENT
- PACKS MICRO

TOUTES NOS OFFRES

- **TABLETTE TACTILE**
  - TOUTES LES TABLETTES
  - KOBO BY
  - MICROSOFT SURFACE
  - IPAD
  - SAMSUNG GALAXY
  - TABLETTE ACER
  - TABLETTE ASUS, NEXUS
  - ACCESSOIRE TABLETTE
- **ORDINATEUR**
  - **ORDINATEUR PORTABLE**
    - Espace

Résultats : **1** - **10** sur un total de **42**

1 - 2 - 3 - 4 - 5 › »

	Modèle	Ecran	Processeur	Système d'exploitation	Note :	Expédié	Prix
BON PLAN 	Tablette Microsoft Surface Pro 3 12" 256 Go - Intel Core i5	12 "	Intel Core i5*	Windows 8 Pro**	4,5/5	En Stock En stock	BON PLAN 1229,90€ ~~1299,90€~~ » Ajouter au panier » 6 neufs à partir de 1229,90€
BON PLAN 	Tablette Microsoft Surface Pro 3 12" 128 Go	12 "	Intel Core i5*	Windows 8 Pro**	4,5/5	En Stock En stock	BON PLAN 929,90€ ~~999,90€~~ » Ajouter au panier » 6 neufs à partir de 929,90€
SOLDE 	Pack Tablette Asus T100TA-DK090H 10.1" Tactile, 1 To + Housse + Microsoft Office 2013	10,1 "	Intel Quad-Core Atom Bay Trail-T	Windows 8 32 Bits**	4/5	En Stock	SOLDE 399,98€ ~~499,90€~~ » Ajouter au panier
BON PLAN 	Tablette Microsoft Surface Pro 3 12" 64 Go	12 "	Intel Core i3*	Windows 8 Pro**	3,5/5	En Stock En stock	BON PLAN 749,90€ ~~799,90€~~ » Ajouter au panier » 5 neufs à partir de 749,90€

UN ALGORITHME DE TRI

Principe du « tri par sélection » (pour trier des entiers par ordre croissant) :

tant que il reste plus d'un élément non trié faire

- **chercher le plus petit parmi les non triés,**
- **échanger le premier élément non trié avec le plus petit élément non trié trouvé.**

fait

Tri par sélection (en Java)

```
triSelection(int[] tab) {
      int n=tab.length, i, j, p, temp;
      for(i=0 ; i<=n-2 ; i++) {
    /* les éléments de 0 à i-1 sont déjà triés
       i : place où on doit mettre le suivant plus petit
       p : place du plus petit des non triés */
            p=i;
            for(j=i+1 ; j<=n-1 ; j++)  {
                  if(tab[j] < tab[p])
                        p=j;
            }
            /* on échange tab[i] et tab[p] */
            temp=tab[i]; tab[i]=tab[p]; tab[p]=temp;
      }
}
```

EXEMPLE

éléments triés éléments non triés				
7	8	15	**5**	10
5	8	15	**7**	10
5	7	15	**8**	10
5	7	8	15	**10**
5	7	8	10	**15**
5	7	8	10	15

Implémentation d'algorithmes :

Passage de paramètres et utilisation de tableaux en Java

Récursivité

Variables et références

- Un programme JAVA utilise des variables pour garder trace des opérations effectuées.
- En fait, le déroulement d'un programme peut se voir comme une suite d'états de la mémoire (vision de la programmation impérative).
- A chaque variable est associée une place mémoire, qui possède une adresse.
- Une ***référence*** vers une variable est la valeur de l'adresse de son emplacement mémoire.

Variables, fonctions et paramètres

- Lors d'un appel de fonction, les valeurs passées en paramètres sont recopiées dans des variables locales (paramètres formels) :

```
public static void f
(int i) {i=5;}
(…) int a=2; f(a); (…)
```

/ Ici, i et a ne référencent pas du tout la même zone mémoire : modifier la valeur de l'un n'a donc aucune influence sur celle de l'autre ! */*

- Ce mécanisme de recopie est **systématique :** quoi qu'un paramètre représente, sa valeur est recopiée dans une variable locale.

Variables de type Objet

- Ce qu'on appelle un *Objet* en Java est en fait *une référence vers un Objet* :

```
String str = new String("oui");//str
   contient l'adresse d'un objet String
```

- Que vaut alors `str` après l'appel suivant ?

```
public static void f2(String s)
{s = new String("non");}
(…) f2(str); (…)
```

Paramètres objets

- Que valent pt.x et pt.y à la fin ?

```
public class Point {public int x=0, y=0;}
(…) public static void changePoint(Point p,
  int x, int y) {p.x=x; p.y=y;} (…)
(…) Point pt=new Point(); (…)
(…) changePoint(pt, 1, 2); (…)
```

Un peu de vocabulaire...

- Dans un langage de programmation, passer un paramètre par référence signifie fournir l'adresse de son emplacement mémoire (et non sa valeur – cf passage par valeur).
- Un objet Java est toujours passé par référence
  - Ainsi, une variable `String str` contient l'adresse d'un objet `String`,
  - Mais cette adresse est recopiée localement lorsque `str` est en paramètre d'une fonction.

Les tableaux

Qu'est-ce qu'un tableau ?

- Un tableau est une structure de données qui réunit un nombre prédéfini de données d'un même type.

- On peut le voir comme une suite de cases contiguës, chacune identifiée (indicée) par un entier.

Spécificités d'un tableau

- La taille (nombre de cases) du tableau doit être connue au moment de sa création et ne peut pas être modifiée. La mémoire nécessaire à toutes les cases est allouée une fois pour toutes.

- On a un accès direct en consultation et en modification à chaque case du tableau (repérée par son indice).

Tableaux en Java (rappels)

- Il existe un type tableau prédéfini en Java.

- Contrairement aux types primitifs (boolean, int, float, etc.), une variable de type tableau :
  - Contient une référence vers un espace mémoire composé d'emplacements mémoire consécutifs.
  - Plus précisément, si on déclare `int[] t`, alors `t` contient l'adresse de la 1ère case du tableau.

Exemples de tableaux en Java

- Exemple :

```
int[] T; /* déclare le tableau d'entiers T et l'initialise à null */
int[] Tbis={1,2,3,4}; /* déclare le tableau Tbis et l'initialise à une référence vers un espace mémoire contenant les 4 entiers 1, 2, 3, 4 */
int[] Tter=new int[4]; /* déclare le tableau d'entiers Tter et l'initialise à une référence vers un espace mémoire contenant 4 entiers */
```

- On peut faire toutes sortes d'opérations, dès lors que le tableau n'est pas vide (`!=null`) :
  - `Tbis[0]=Tbis[0]+10;`
  -
```
for(int i=0; i<Tbis.length; i++)
        System.out.println(Tbis[i]);
```

Tableaux d'objets en Java

- Attention :
  - L'instruction `new` permet d'allouer de la place pour un certain nombre de références vers des objets, mais...
  - De l'espace mémoire doit ensuite être alloué pour chaque objet référencé par le tableau !

- Par exemple :
  - `String[] Tab=new String[3];`
  - `if(Tab[0].equals("toto")) {return true;}`
  - Ce bout de code produit une exception (le programme s'arrête alors), car `Tab[0]` vaut `null` !
  - Il manque : `Tab[0]=new String();`

Tableaux et références

- On déclare `int[] tab=new int[6];`
  - Puis, on écrit :

```
(…) public static void f3(int[] t)
  {for(int i=0;i<6;i++) t[i]=i+1;} (…)
(…) f3(tab); (…) /* que vaut tab ? */
```

  - On écrit ensuite :

```
(…) public static void f4(int[] t)
  {t=new int[1]; t[0]=9;} (…)
(…) f4(tab); (…) /* que vaut tab ? */
```

Récursivité

- On dit qu'une fonction est récursive lorsqu'elle s'appelle elle-même (en général, avec un jeu de paramètres différent).

- La récursivité est un concept puissant, qui permet de traduire naturellement (et parfois plus simplement que par une syntaxe itérative) les processus de nature « récurrente ». Ex. :

```
public static int factorielle(int n) {
   if(n>1) return(n*factorielle(n-1));
   else return 1;
}
```

« Dangers » de la récursivité

- Faire des appels récursifs « en cascade » peut amener à en générer un très grand nombre !
  - Ainsi, calculer le déterminant d'une matrice de cette façon-là génère un nombre exponentiel d'appels...

- A ***chaque*** appel, on recopie les paramètres dans des variables locales : un algorithme récursif consomme donc parfois beaucoup plus de mémoire que son équivalent itératif...

BILAN

- Pour être capable de résoudre un problème avec un ordinateur, il faut avoir compris :
  - Comment on peut résoudre le problème, indépendamment du langage et de l'ordinateur qui sera utilisé (algorithme),
  - Comment traduire cet algorithme en langage informatique, pour obtenir un programme,
  - Comment le programme sera exécuté par l'ordinateur (gestion des instructions et de la mémoire) ==> ***prochaines séances de cours***