Chapitre 0: Introduction à l'algorithmique

Brice Mayag
brice.mayag@dauphine.fr

M1 SIREN: maj. SIEE/GSI

Dauphine | PSL
UNIVERSITÉ PARIS

Sommaire

Sommaire

Objectif du cours

Sept séances

- En “présentiel”

Les grandes notions

- Les bases de l'algorithmique et de la programmation
- Structures de données basiques et avancées : listes, files, arbres de recherche
- Implémentation et tests avec le langage Python
- Des exemples et application à quelques cas simples

Sommaire

L'algorithmique ?

Définition (informelle)

Un algorithme est la composition d'un ensemble fini d'étapes, chaque étape étant formée d'un nombre fini d'opérations dont chacune est :

- définie de façon rigoureuse et non ambigue ;
- effective (i.e. pouvant être réalisée en un temps fini).

La notion d'algorithme est plus générale que celle de programme (indépendant du langage de programmation utilisé).

Un peu d'histoire

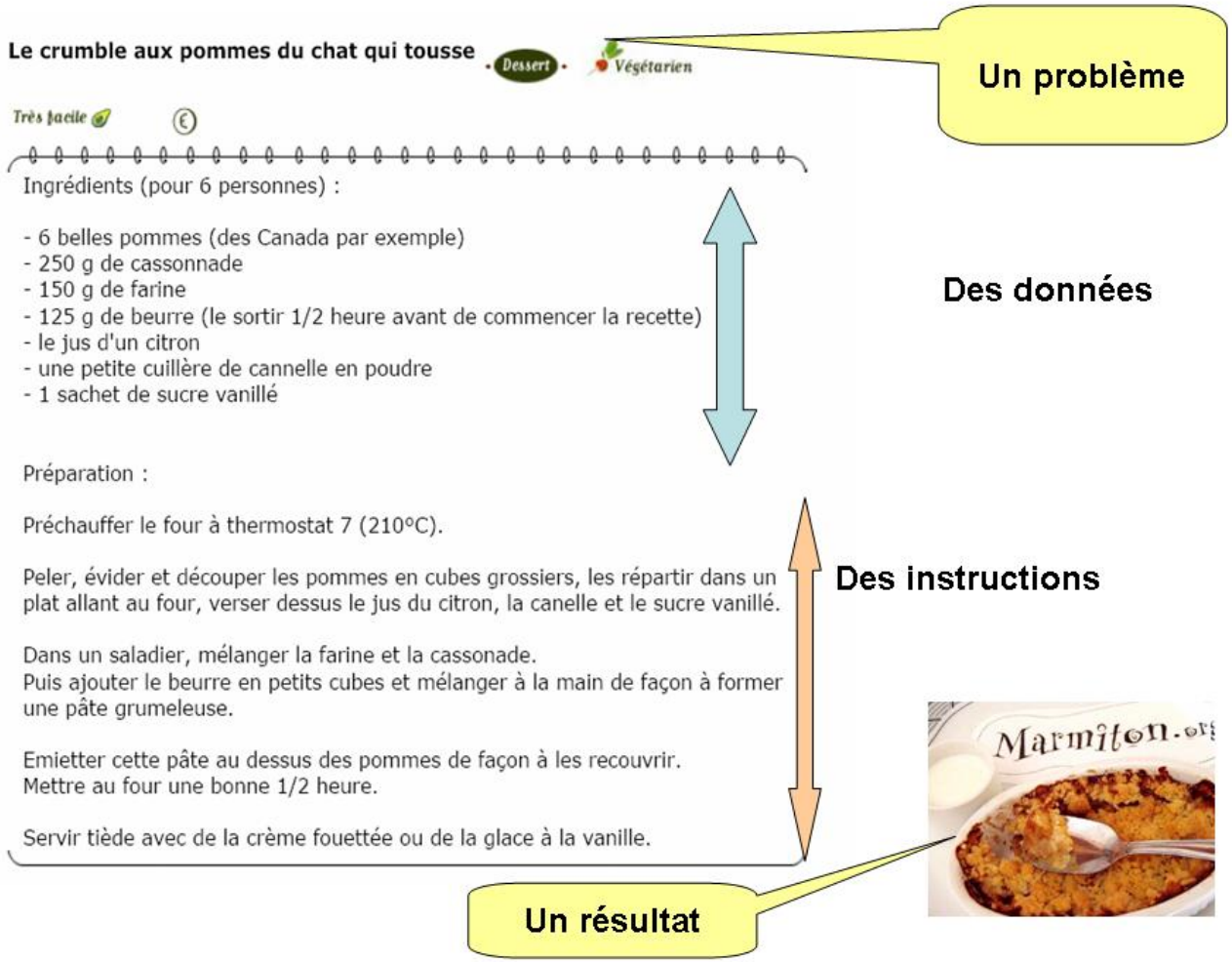
Le mot algorithme vient du nom du mathématicien Al Khwarizmi (820) : introduction de la numérotation décimale et des calculs s'y rapportant.

Des besoins contradictoires

Un algorithme doit :

- être simple à comprendre, à mettre en oeuvre et à mettre au point ;
- mettre intelligement à contribution les ressources de l'ordinateur, et plus précisément, il doit s'exécuter rapidement.

Un algorithme : c'est quoi ?



Classiquement assimilable à une recette de cuisine

Algorithme : définitions

Un algorithme =

- Description précise des opérations à faire pour résoudre un problème (suite d'instructions).
- Procédure de calcul bien définie qui prend en entrée une valeur, ou un ensemble de valeurs et qui donne en sortie une valeur, ou un ensemble de valeurs.

Un *bon* algorithme =

- Un algorithme correct : i.e. pour chaque instance en entrée, l'algorithme se termine en produisant la bonne sortie
  $\Rightarrow$ Savoir prouver un algorithme
- Un algorithme efficace : mesure de la durée que met un algorithme pour produire un résultat
  $\Rightarrow$ Savoir analyser la complexité d'un algorithme : i.e. détermination de l'espace mémoire et du temps d'exécution nécéssaire à la résolution du problème.

Pour un problème donné, plusieurs algorithmes ou aucun sont possibles.
Un algorithme se termine en un temps fini.

Dauphine | PSL UNIVERSITÉ PARIS

En résumé . . .

Méthode et developpement d'un algorithme

1. Spécification du problème (énoncé)
2. Conception préliminaire
3. Conception détaillée
4. Implémentation (ne doit pas obscurcir les étapes précédentes !)

- Les étapes 2 et 3 relèvent de l'algorithmique
- L'étape 4 relève de la programmation
- Avant de passer à la programmation et à la manipulation machine, voyons un exemple plus détaillé de problème algorithmique !

Un exemple simple : le problème de tri

Problème de tri

- Entrée : une suite de n nombres, $e_0, e_1, \ldots, e_{n-1}$
- Sortie : la même suite de nombre, mais ordonnée par ordre croissant (ou décroissant)

  i.e., permutation de la suite donnée en entrée $(e'_0, e'_1, ..., e'_{n-1})$ de telle sorte que $e'_0 \leq e'_1 \leq ... \leq e'_{n-1}$

- Plusieurs stratégies (et donc algorithmes) sont possibles, avec des complexités différentes !
- Tri par sélection
- Tri par insertion
- Tri à bulles
- Tri fusion
- ...

Un exemple simple : le tri par sélection

Principe du tri par sélection

- Rechercher le plus grand élément (ou le plus petit), le placer en fin de tableau (ou en début).
- Recommencer avec le second plus grand (ou le second plus petit), le placer en avant-dernière position (ou en seconde position) et ainsi de suite jusqu'à avoir parcouru la totalité du tableau

Un exemple simple : tri par sélection

Principe

On veut trier, du plus petit au plus grand, la liste : | 3 | 2 | 7 | 1 | 5 |

1. L'élément le plus petit se trouve en 3ème position (si on commence à compter à partir de zéro) : | 3 | 2 | 7 | 1 | 5 |
2. On échange l'élément le plus petit (en 3ème position) avec le premier :

1	2	7	3	5

3. On continue en considérant le même tableau, et en ignorant son premier élément (qui est déjà trié)
4. | 1 | 2 | 7 | 3 | 5 |
|---|---|---|---|---|
5. Et ainsi de suite, en ignorant à chaque fois les éléments déjà triés (en rouge).

Un exemple simple : tri par sélection

```
tri.selection(x, n)
for i from 1 to n-1 do
    min ← i
    for j from i + 1 to n do
        if x[j] < x[min] then
            min ← j
        end if
    end for
    if min ≠ i then
        Switch x[i] and x[min].
    end if
end for
```

Nous l'implémenterons en Python en TP.

Un exemple simple : tri par sélection

Combien d'opérations sont nécessaires ?

- Pour trouver l'élément le plus petit, l'algorithme doit effectuer $n-1$ comparaisons.
- Pour trouver le deuxième élément le plus petit, l'algorithme doit effectuer $n-2$ comparaisons.
- ... Au total, le nombre d'opérations nécessaires est

$$(n-1)+(n-2)+\ldots+1=\frac{n(n-1)}{2}$$

- On dit que la complexité est de l'ordre de $\mathcal{O}\left(n^2\right)$
- Une complexité quadratique n'est généralement pas la plus efficace pour un algorithme de tri

Complexité d'un algorithme

Permet de quantifier les algorithmes

Deux types de compléxité :

- **En espace** : Quelle quantité de place en mémoire va t-on utiliser ?
- **En temps** : Combien d'opérations va t-on effectuer ?

Intérêt : comparer deux algorithmes

Mesurer le temps d'exécution

- Le temps d'exécution dépend de l'entrée : par exemple dans le cas d'un algorithme de tri, si le tableau est déja trié.
- On cherche une fonction $T(n)$ représentant le **temps d'exécution** d'un algorithme en fonction de la **taille de l'entrée** n (nombre d'élements constituant l'entrée, nombre de bits nécéssaire à la représentation de l'entrée,...)
- Le calcul exact étant souvent impossible à appréhender exactement, on s'intéresse aux :
  - Meilleur des cas
  - Pire de cas
  - Cas moyen : nécéssite d'avoir des connaissances sur la distribution statistique des entrées

Classes de complexité

- Algorithmes sub-linéaires : $\Theta(\log n)$
- Algorithmes linéaires : $\Theta(n)$ et $\Theta(n \log n)$
- Algorithmes polynomiaux : $\Theta(n^k)$
- Algorithmes exponentiels : $\Theta(2^n)$

La question de la complexité ne sera pas discutée en détail dans ce cours, mais il faut toujours l'avoir en tête, et notamment lorsqu'on doit traiter des données massives, etc.

Sommaire

Comment "parler" à un ordinateur ?

Le langage machine

- Dans son fonctionnement interne, un ordinateur n'est capable de traiter autre chose que des nombres binaires
- Toute information doit être traduite ou codée en format binaire
- Cela est vrai pour les données (nombres, images, textes, sons, ...), mais aussi pour les programmes, c'est à dire pour la suite d'instruction que l'on va fournir à la machine pour lui dire ce qu'elle doit faire.
- Le langage machine est donc incompréhensible pour l'être humain et il faut des systèmes de traduction automatiques comme un interpréteur ou un compilateur.

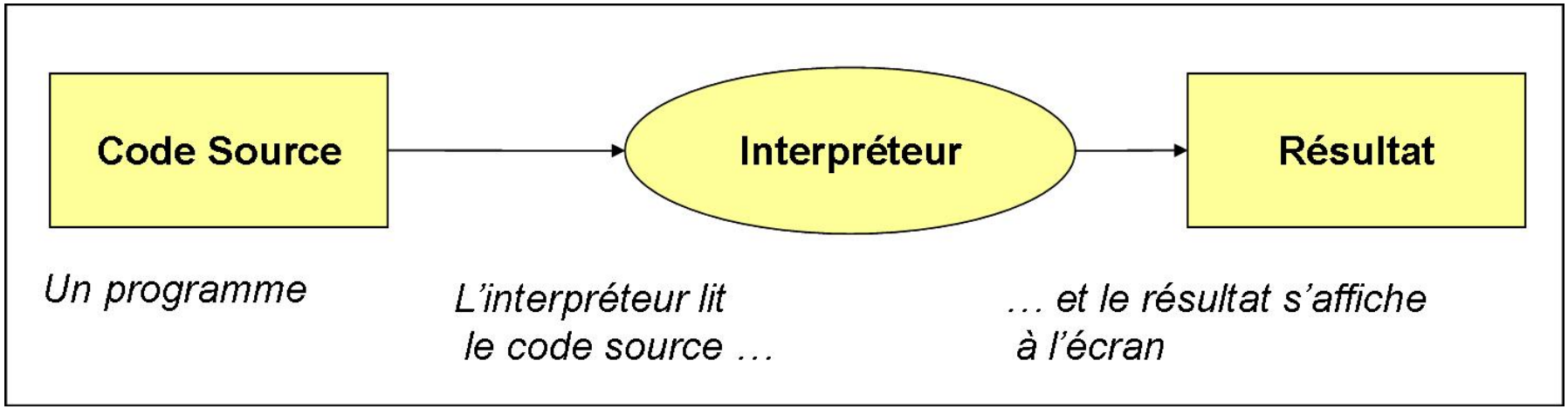
Comment "parler" à un ordinateur ?

Langages de programmation

- Ensembles de mots clefs et de règles pour former des phrases pouvant être traduites par la machine (binaire).
- Le langage de programmation est l'outil qui permet de traduire un algorithme de manière formelle pour que l'ordinateur puisse le comprendre et l'exécuter.
- Plusieurs niveaux :
  - Langages de bas niveau : instructions élémentaires très *proches de la machine* (ex : Assembleur : V AX, [0110] signifie *copier le contenu de 0110 dans le registre AX* ; langage machine)
  - Langages de haut niveau : instructions plus abstraites (C, C++, Perl, Java, Python).

Production de programmes : Interprétation

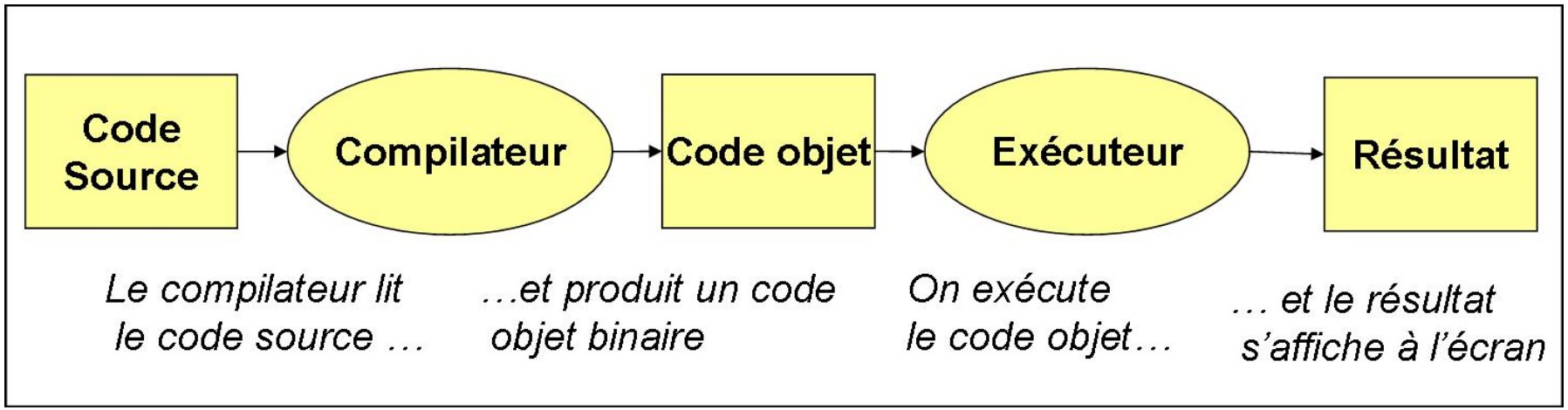
Interprétation



Chaque ligne du code source est traduite au fur et à mesure en instructions qui sont directement exécutées, i.e. l'interpréteur est utilisé à chaque exécution.

Production de programmes : Compilation

Compilation



Compilation = traduction du code ecrit dans un langage dit source en langage objet ou cible (analyse lexicale, syntaxique, sémantique et production du code objet). L'édition de liens ..

Qu'est-ce qu'un programme

Programme

Une suite d'instructions qui spécifient comment exécuter une tâche ou résoudre un problème.

Eléménts d'un programme

Alors qu'il existe des différences entre les différents langages de programmation, il existe toujours les mêmes éléments communs dans un programme

- Un input ou une entrée
- Un output ou une sortie
- Un peu de mathématiques
- Instructions conditionnelles
- Des boucles

Sommaire

Le langage Python

C'est

- Un langage de scripts (de petits programmes très simples chargés d'une mission très précise sur votre ordinateur) ;
- Un langage impératif interprété (c'est-à-dire que les instructions que vous lui envoyez sont "transcrites" en langage machine au fur et à mesure de leur lecture) contrairement à un langage compilé (C/C++)

Le langage Python

Python est un langage de programmation

Ce qui définit un langage de programmation

- La façon de représenter symboliquement les structures de données.
- La façon de gérer le contrôle des programmes (que faire et dans quel ordre)

Comme tout langage de programmation, Python fournit un certain nombre de types de données prédéfinis (booléens, entiers, flottants, chaîne de caractères, listes) et de structure de contrôle (les boucles for, while et les conditions if_then_else_).

Installation et Prise en main

Dans ce cours, on pourra utiliser la distribution Anaconda, `https://www.anaconda.com`, qui propose une intégration directe des principaux packages, ainsi qu'un environnement de développement intégré appelé Spyder.

Programme

- Introduction à l'algorithmique
- Variables, types et boucles
- Fonctions
- Dictionnaires
- Applications : Filtrage collaboratif (systèmes de recommandation)
- ...

Evaluation

- Mini projet à faire en binôme, à la maison (50%)
- Examen en binôme, à la dernière séance (50%)