cours_1

January 14, 2020

Algorithmique des tableaux
Christophe Saint-Jean
Transparents du cours
Code du cours
Année 2019-2020

1 Organisation de l'UE

1.1 Mentions Légales

1.2 L'équipe enseignante

- **Christophe Saint-Jean** (CM/TD/TP - Resp.)
- Laurent Mascarilla (TD/TP)
- El Hadi Zahzah (TD/TP)

1.3 Communication

- Questions pédagogiques : Moodle
  - Appronfondissement/Questions (Forum)
  - Organisation de l'UE/Planning (Messages privés)
- Questions administratives (Secrétariat)
  - Appartenance groupes TD/TP
  - Absences/Justifications

1.4 Dispositif horaire

- 6 cours de 1,5 heures (Amphithéâtre)
- 4 TDs de 1,5 heures
- 5 TPs de 1,5 heures (Sallles de TP)
- 4 créneaux de 1,5h de TEA - 2 sujets (Salles de TP)

1.5 Evaluation

$$S_1 = \frac{CC_1 + CC_2}{2}$$

$$S_2 = CC_3$$

Les CC se passent *a priori* en TP 3 et 5 sur machine. Attention à la règle sur les absences.

1.6 Les objectifs de cet enseignement

- Initiation à l'algorithmique
  - Qu'est ce qu'un algorithme ?
  - Différence algorithme / programme (cf. TD)
  - Initiation à l'analyse d'un algorithme.
- Approfondir vos connaissances sur :
  - les listes, tableaux, fonctions.
  - le langage Python.
- Découvrir des algorithmes simples et les analyser.
- Initiation à la récursivité.

2 Généralités sur l'algorithmique

2.1 Analogie algorithme/recette

1. Mettez la farine dans un saladier avec le sel et le sucre.
2. Faites un puits au milieu et versez-y les ufs légèrement battus à la fourchette.
3. Commencez à incorporer doucement la farine avec une cuillère en bois. Quand le mélange devient épais, ajoutez le lait froid petit à petit.
4. Quand tout le lait est mélangé, la pâte doit être assez fluide, si elle vous paraît trop épaisse, rajoutez un peu de lait. Ajoutez ensuite le beurre fondu, mélangez bien.
5. Faites cuire les crêpes dans une poêle chaude.
6. Répétez jusqu'à épuisement de la pâte.

On attend:

- Langage de description intelligible
- Instructions séquentielles, fonctions, répétitives.

On distingue bien:

- la recette (l'algorithme)
- de sa mise en oeuvre (traduction dans un langage de programmation =: Implémentation)
- aux moyens d'ustentiles de cuisine (variables, structures de données, fonctions, ...)

2.2 Algorithme (Définition)

Une définition formelle:

`Un algorithme est la description dune méthode de calcul qui, à partir dun ensemble de données dentrée (problème) et une suite finie d'étapes, produit un ensemble de données en sortie (solution).`

`Un algorithme est la description dune méthode de calcul ...`

Description

On doit décrire chaque fonction (sous-algorithmes) non triviale et structures de données employées

Méthode de calcul

Il ne peut résoudre que des problèmes calculables. On démontre que certains problèmes ne sont pas calculables (décidables).

Ex.: Problème de l'arrêt

`Un algorithme est .... et une suite finie d'étapes, produit un ensemble de données en sortie (solution).`

Une suite finie d'étapes

Attention, ne pas confondre:

- Une séquence d'instructions qui se termine.
- Une description de longueur finie possibilité de boucle infinie.

Solution

L'algorithme apporte t'il une solution au problème posé ?

2.3 Algorithmique (Définition)

`L'algorithmique est la science qui étudie les algorithmes pour eux-même indépendamment de tout langage de programmation.`

d'après "Al Khwarizmi", surnom du mathématicien arabe Muhammad Ibn Musa (IX siècle).

2.3.1 Algorithmique (Histoire)

L'algorithmique et les algorithmes sont bien antérieurs à l'informatique:

- Abaques grecques, romaines, PGCD d'Euclide (IIIème siècle av. J.-C.)
- Boulier chinois (XIIIème siècle)
- Pascaline (1646)

2.3.2 Questions de l'algorithmique

1. L'algorithme A est t'il correct ?
2. L'algorithme A se termine t'il ?
3. L'algorithme A est t'il plus efficace qu'un algorithme B ?

Parallèlement, des questions plus fondamentales:

- Est il possible de trouver un algorithme qui résoud un problème P ? (Décidabilité)
- Si, oui existe t'il un algorithme efficace résoud un problème P ? (Classes de complexité)

2.3.3 Exemples d'algorithmes (cf. S1)

Algorithme 1: Plus grand élément d'un tableau

Données : T un tableau n de valeurs comparables par $<$
Résultat : La plus grande valeur de T

```
1 max ← T[1];
2 pour i ← 2 à n faire
3 |   si max < T[i] alors
4 |   |   max ← T[i];
5 |   fin
6 fin
7 retourner max
```

max

Algorithme 2: Approximation de π

Données : π_c une valeur approchée précise de π, p un entier positif
Résultat : Une approximation $\hat{\pi}$ de π_c à 10^{-p} près

```
1 som ← 0 ;                                  // Initialiser som à 0
2 π̂ ← 0 ;                                    // Initialiser π̂ à 0
3 tant que |π̂ − π_c| ≥ 10^{-p} faire
4 |   som ← som + (−3^k)/(2k+1) ;             // Ajouter (−3^k)/(2k+1) à som
5 |   π̂ ← √12 som;
6 |   k ← k + 1 ;                             // Incrémenter k
7 fin
8 retourner π̂
```

approxpi

2.4 Description d'un algorithme

On utilisera un langage de description d'un algorithme appelé **pseudo-code**.

Caractérisques du pseudo-code

- Il ne doit pas être attaché la syntaxe d'un langage informatique particulier.
- Il doit être lisible par un non-programmeur.
- Être capable de décrire les structures de contrôle des langages impératifs (If, While, For, ...).

2.4.1 L'interface de l'algorithme

Quelques sont les entrées attendues par l'algorithme ?

- Type : Nombre, Tableau, Liste, Arbre, etc ...
- Taille : nombre de bits, nombre d'éléments du tableau, nombre de feuilles, ...
- Propriétés : entiers positifs, tableau trié, ...

Que fait/produit l'algorithme ?

- *Idem* que sur les entrées
- Description textuelle (éventuelle) de l'algorithme

2.4.2 Les types utilisables

Types de données élémentaires :

- Variables simples : booléen, entier, réel, caractère
- Tableaux
- *Pointeurs*

Cela permet de définir des structures de plus haut niveau:

- Chaîne de caractères
- Ensemble, Collection
- Liste, *table de hachage*
- *Graphes, . . .*

2.4.3 Quelques instructions du pseudo-code

- Affectation : <-
- Test : =
- Opérations arithm etiques : +,-,*,/
- Séparateur d'instructions : " ;"
- Elements d'un tableau T : "T[i]" (convention 1..n !!)
- Adresse d'une variable "@"
- Instruction de retour : Retourner <val. sortie>

- Le branchement conditionnel :

```
Si <condition> alors
  <blocsi>
sinon
  <blocsinon>
fin
```

- Les itératives et les répétitives:

```
Pour i <- 1 à n [par pas de 1] faire
    <bloc>
fin

Tant que <condition> faire
    <bloc>
fin
```

2.4.4 Langage de description et Python

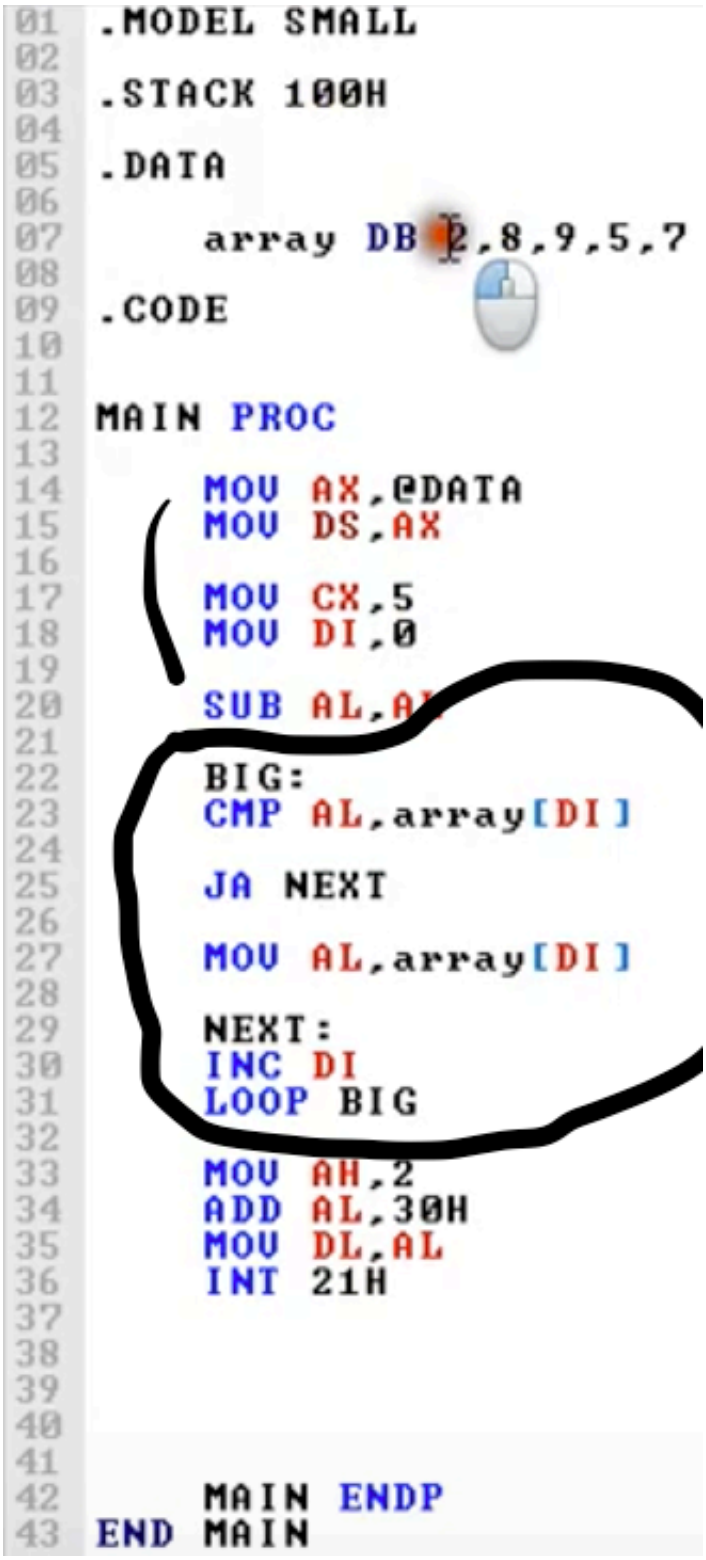
- Le langage Python a été conçu pour être le plus lisible et naturel possible.
- On est très proche du langage de description (raccourci en TD)
- **Un algorithme devient une fonction !!**

```
def max(T):
    maxi = T[0]
    for Ti in T[1:]:
        if maxi < Ti:
            maxi = Ti
    return maxi
```

En C++

```
template<class T> T max(const T* data, int size) {
    T result = data[0];
    for(int i = 1 ; i < size ; i++)
      if(result < data[i])
        result = data[i];
    return result;
  }
```

En Assembleur



```asm
.MODEL SMALL

.STACK 100H

.DATA

    array DB 2,8,9,5,7

.CODE


MAIN PROC

    MOV AX,@DATA
    MOV DS,AX

    MOV CX,5
    MOV DI,0

    SUB AL,AL

    BIG:
    CMP AL,array[DI]

    JA NEXT

    MOV AL,array[DI]

    NEXT:
    INC DI
    LOOP BIG

    MOV AH,2
    ADD AL,30H
    MOV DL,AL
    INT 21H



    MAIN ENDP
END MAIN
```

asm

2.4.5 De l'importance d'une bonne description

Algorithme 3: Calcul de l'élément maximal d'un tableau trié

Données : T un tableau trié dans l'ordre croissant de n entiers
Résultat : $i_{\max}$ et max tel que $T[i_{\max}] = \max$ avec max comme la plus grande valeur de T

```
1 i_max ← n;
2 max ← T[n];
```

Maximum Trié

2.5 Analyse: Preuve de terminaison

- Vérifier que chaque instruction simple se termine:
  - calcul simple, affectation OK
  - affichage OK
  - Appel de fonction -> vérifier la fonction
- Instruction Si : vérifier la condition et les deux branches possibles
- Pour les boucles for, s'assurer que la séquence parcourue est taille finie.
- Pour la répétitive While, s'assurer que dans tous les cas que la condition de continuation sera fausse au moins une fois.
- Pour les algorithmes récursifs (plus tard), on doit s'assurer que la récursion se termine

Algorithme 1: Plus grand élément d'un tableau

Données : T un tableau n de valeurs comparables par $<$
Résultat : La plus grande valeur de T

```
1 max ← T[1];
2 pour i ← 2 à n faire
3 |   si max < T[i] alors
4 |   |   max ← T[i];
5 |   fin
6 fin
7 retourner max
```

max

2.6 Analyse: Preuve de correction

Il est question de prouver que l'algorithme fait ce qu'il dit faire !

On utilise souvent un invariant de boucle et la preuve par récurrence.

Pour des algorithmes concernant un tableau T[1..n], la démarche (simplifiée) est généralement la suivante:

- Soit i l'indice du parcours de T (c.a.d. T[i])
- Avant l'itération, quelle propriété vérifie la variable qui sera retournée par rapport à T[1..i-1] ?
- Après l'itération, cette propriété reste t'elle vraie pour T[1..i] ?
- L'initialisation de la variable est elle compatible à la propriété ?

On a donc les deux éléments d'une récurrence: Initialisation et Hérédité.

Algorithme 1: Plus grand élément d'un tableau

Données : T un tableau n de valeurs comparables par $<$
Résultat : La plus grande valeur de T

```
1 max ← T[1];
2 pour i ← 2 à n faire
3 |   si max < T[i] alors
4 |   |   max ← T[i];
5 |   fin
6 fin
7 retourner max
```

max

2.7 Analyse : Complexité algorithmique

L'algorithme est il rapide ?
Pour un tableau T de taille n, la rapidité *devrait* dépendre de:

- n la taille T.
- d'une propriété, du contenu de T.
- du langage de programmation ?

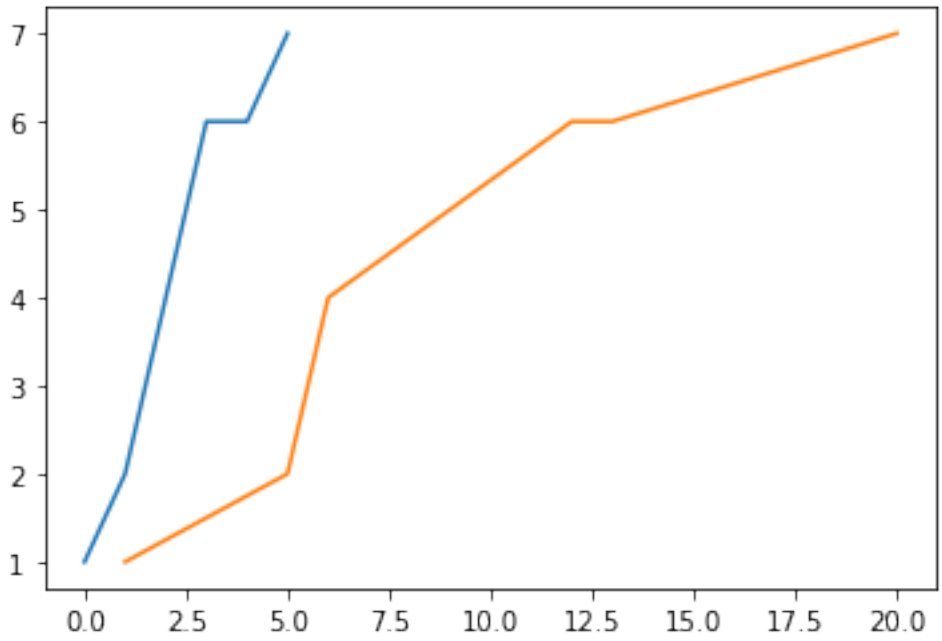
Les outils du jour:

- Mesurer le temps d'exécution du programme implémentant l'algorithme (module time)
- Tracer une courbe (module matplotlib)
- Le module tqdm

[1]:
```
from time import time, sleep

debut = time()
###    Votre code ici
sleep(2)
fin = time()
duree = fin - debut
print("Duree: ", duree)
```

```
Duree:  2.0020060539245605
```

[2]:
```
import matplotlib.pyplot as plt # a installer

y = [1, 2, 4, 6, 6, 7]
x1 = list(range(len(y)))
x2 = [1, 5, 6, 12, 13, 20]

plt.plot(x1, y)
#plt.show()
plt.plot(x2, y)
plt.show()
```



[3]:
```
# tqdm a installer
#from tqdm import tqdm                    # programme python classique
from tqdm.notebook import tqdm           # dans un notebook
#from tqdm.gui import tqdm                # pour thonny
from time import sleep
from random import randint

for i in tqdm(range(3), desc='Boucle sur i'):
    for j in tqdm(range(4), desc=f'Boucle sur j', leave=True):
        duree_sommeil = randint(1, 4)
        sleep(duree_sommeil)
```

```
HBox(children=(FloatProgress(value=0.0, description='Boucle sur i', max=3.0, style=ProgressSty
```

```
HBox(children=(FloatProgress(value=0.0, description='Boucle sur j', max=4.0, style=ProgressSty
```

```
HBox(children=(FloatProgress(value=0.0, description='Boucle sur j', max=4.0, style=ProgressSty
```

```
HBox(children=(FloatProgress(value=0.0, description='Boucle sur j', max=4.0, style=ProgressSty
```