

Module EA4 – Éléments d'Algorithmique II

Outils pour l'analyse des algorithmes

Dominique Poulalhon

`dominique.poulalhon@irif.fr`

Université Paris Diderot

L2 Informatique & DL Bio-Info, Jap-Info, Math-Info

Année universitaire 2020-2021

ORGANISATION DU MODULE

Emploi du temps (en théorie)

- Cours : 2h par semaine, mercredi 13h45-15h45, amphi 1A
- TD : 2h par semaine
- TP : 2h par quinzaine (en alternance, pour une question de gestion des salles du Script)

Emploi du temps (en pratique – pour le moment en tout cas...)

- CM sur **BBB** à l'horaire prévu
- TD sur **discord** à l'horaire prévu : accueil de chaque groupe dans un couple de salons textuel+vocal dédié, puis travail en petits groupes dans les mini-salons
- TP : en autonomie, avec une permanence des enseignants sur **discord** pendant toute la journée du jeudi (à peu près par quinzaine, même semaine pour tous les groupes)

ÉQUIPE ENSEIGNANTE

Responsable du cours : Dominique Poulalhon

`dominique.poulalhon@irif.fr`

Chargés de TD-TP

- Groupe INFO 1 : Roberto Mantaci `mantaci@irif.fr`
- Groupe INFO 2 : Vincent Cheval `vincent.cheval@inria.fr`
- Groupe INFO 3 : Maxime Bombar `maxime.bombar@inria.fr`
- Groupe INFO 4 : Dominique Poulalhon
- Groupe INFO 5 : Matthieu Picantin `picantin@irif.fr`
- Groupe INFO 6 : Giovanni Bernardi `gio@irif.fr`
- Groupe MI 1 : Camille Combe `combe@irif.fr`
- Groupe MI 2 : Anne Micheli `anne.micheli@irif.fr`

COMMUNICATION

Un site Moodle pour les annonces, les supports de cours, les énoncés, les rendus de TP...

- inscrivez-vous dans le bon groupe
- attention à bien utiliser votre compte u-paris.fr (avec lequel vous êtes censés avoir été automatiquement inscrits) et *pas* univ-paris-diderot.fr, qui est voué à disparaître

Un serveur discord – des salons pour les TD, les TP, et si vous le souhaitez des messages privés ; mais dans ce cas *toujours mentionner à propos de quel cours vous nous écrivez*, et précisez votre nom si votre pseudo usuel n'est pas transparent

Quelques règles sur le serveur : adoptez un pseudo qui permette de vous identifier ; pensez à couper le micro lorsque vous ne parlez pas – mais n'hésitez surtout pas à intervenir !

Et bien sûr le mail – toujours mentionner [EA4] dans le sujet

MODALITÉS DE CONTRÔLE DES CONNAISSANCES

Session 1 : Contrôle Continu Intégral

Contenu à définir, mais probablement :

- un contrôle final sur table qui comptera pour 50%,
- si possible un autre à mi-semestre,
- une ou des évaluation(s) via moodle,
- et une note de participation basée sur... la participation et les rendus de TP

Session 2 : Examen

THÈME DU COURS

algorithmique = « conception et analyse des algorithmes »

THÈME DU COURS

algorithmique = « conception et analyse des *algorithmes* »

THÈME DU COURS

algorithmique = « conception et analyse des *algorithmes* »

algorithme = « méthode (systématique) de résolution d'un problème »

THÈME DU COURS

algorithmique = « conception et analyse des *algorithmes* »

algorithme = « méthode (systématique) de résolution d'un problème »

concept non limité à l'informatique – d'ailleurs, de nombreux algorithmes ont été décrits bien avant l'invention des ordinateurs :

- des algorithmes de calcul
(opérations arithmétiques, approximation de π , de $\sqrt{2}$...)

THÈME DU COURS

algorithmique = « conception et analyse des *algorithmes* »

algorithme = « méthode (systématique) de résolution d'un problème »

concept non limité à l'informatique – d'ailleurs, de nombreux algorithmes ont été décrits bien avant l'invention des ordinateurs :

- des algorithmes de calcul
(opérations arithmétiques, approximation de π , de $\sqrt{2}$...)
- des constructions géométriques
(milieu d'un segment, triangle équilatéral, droites parallèles,
centre d'un cercle, pentagone régulier...)

THÈME DU COURS

algorithmique = « conception et analyse des *algorithmes* »

algorithme = « méthode (systématique) de résolution d'un problème »

concept non limité à l'informatique – d'ailleurs, de nombreux algorithmes ont été décrits bien avant l'invention des ordinateurs :

- des algorithmes de calcul
(opérations arithmétiques, approximation de π , de $\sqrt{2}$...)
- des constructions géométriques
(milieu d'un segment, triangle équilatéral, droites parallèles, centre d'un cercle, pentagone régulier...)
- des recettes de cuisine



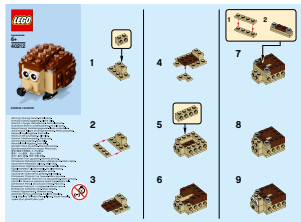
THÈME DU COURS

algorithmique = « conception et analyse des *algorithmes* »

algorithme = « méthode (systématique) de résolution d'un problème »

concept non limité à l'informatique – d'ailleurs, de nombreux algorithmes ont été décrits bien avant l'invention des ordinateurs :

- des algorithmes de calcul
(opérations arithmétiques, approximation de π , de $\sqrt{2}$...)
- des constructions géométriques
(milieu d'un segment, triangle équilatéral, droites parallèles, centre d'un cercle, pentagone régulier...)
- des recettes de cuisine
- des manuels de construction...



THÈME DU COURS

algorithmique = « conception et analyse des *algorithmes* »

algorithme = « méthode (systématique) de résolution d'un problème »

concept non limité à l'informatique – d'ailleurs, de nombreux algorithmes ont été décrits bien avant l'invention des ordinateurs :

- des algorithmes de calcul
(opérations arithmétiques, approximation de π , de $\sqrt{2}$...)
- des constructions géométriques
(milieu d'un segment, triangle équilatéral, droites parallèles,
centre d'un cercle, pentagone régulier...)
- des recettes de cuisine
- des manuels de construction...

mais le concept a pris une importance particulière avec l'apparition de machines capables d'exécuter *fidèlement* et *rapidement* une suite d'opérations prédéfinie

THÈME DU COURS

algorithmique = « conception et analyse des algorithmes »

algorithme = « méthode (systématique) de *résolution* d'un problème »

Étymologie : Muhammad Ibn Mūsā al-Khuwārizmī

mathématicien persan du début du 9^e siècle

« *Kitābu 'l-mukhtasar fī hisābi 'l-jabr wa'l-muqālah* » ou « *Abrégé du calcul par la restauration et la comparaison* » : considéré comme le premier manuel d'algèbre, explique comment résoudre les équations du second degré

traduit en latin et diffusé en Europe à partir du 12^e siècle

(c'est aussi grâce à un de ses livres que se répand la notation positionnelle décimale venue d'Inde)

le terme *algorithme* est d'abord utilisé pour désigner les méthodes (de calcul) utilisant des chiffres, par opposition au *calcul* traditionnel (du latin *calculus*, petit caillou) avec des abaques

THÈME DU COURS

algorithmique = « conception et analyse des algorithmes »

algorithme = « méthode (systématique) de *résolution* d'un problème »

THÈME DU COURS

algorithmique = « conception et analyse des algorithmes »

algorithme = « méthode (systématique) de *résolution* d'un problème »

trois axes d'étude :

- conception
- preuve de correction
- étude de l'efficacité

THÈME DU COURS

algorithmique = « conception et analyse des algorithmes »

algorithme = « méthode (systématique) de *résolution* d'un problème »

trois axes d'étude :

- conception
- preuve de correction : un algorithme est *correct* si, pour chaque entrée, il *termine* en produisant la *bonne sortie*
- étude de l'efficacité

THÈME DU COURS

algorithmique = « conception et analyse des algorithmes »

algorithme = « méthode (systématique) de *résolution* d'un problème »

trois axes d'étude :

- conception
- preuve de correction : un algorithme est *correct* si, pour chaque entrée, il *termine* en produisant la *bonne sortie*
- étude de l'efficacité : les ressources nécessaires (temps, mémoire) sont-elles raisonnables ? Est-il possible de faire mieux ?

THÈME DU COURS

algorithmique = « conception et analyse des algorithmes »

algorithme = « méthode (systématique) de *résolution* d'un problème »

trois axes d'étude :

- conception : y a-t-il des *techniques générales* ?
- preuve de correction : un algorithme est *correct* si, pour chaque entrée, il *termine* en produisant la *bonne sortie*
- étude de l'efficacité : les ressources nécessaires (temps, mémoire) sont-elles raisonnables ? Est-il possible de faire mieux ?

THÈME DU COURS

algorithmique = « conception et analyse des algorithmes »

algorithme = « méthode (systématique) de *résolution* d'un problème »

trois axes d'étude :

- conception : y a-t-il des *techniques générales* ?
- preuve de correction : un algorithme est *correct* si, pour chaque entrée, il *termine* en produisant la *bonne sortie*
- étude de l'efficacité : les ressources nécessaires (temps, mémoire) sont-elles raisonnables ? Est-il possible de faire mieux ?

(et au passage, on apprendra un peu de **PYTHON**, parce que c'est un joli langage particulièrement adapté à l'algorithmique)

EXEMPLE : LES OPÉRATIONS ARITHMÉTIQUES

Addition de deux entiers :

$$\begin{array}{rcccc} & 1 & 3 & 5 & 7 \\ + & 2 & 4 & 6 & 8 \\ \hline \end{array}$$

EXEMPLE : LES OPÉRATIONS ARITHMÉTIQUES

Addition de deux entiers :

$$\begin{array}{rcccc} & & & 1 & \\ & 1 & 3 & 5 & 7 \\ + & 2 & 4 & 6 & 8 \\ \hline & & & & 5 \end{array}$$

EXEMPLE : LES OPÉRATIONS ARITHMÉTIQUES

Addition de deux entiers :

$$\begin{array}{rcccc} & & 1 & & 1 \\ & 1 & 3 & 5 & 7 \\ + & 2 & 4 & 6 & 8 \\ \hline & & & 2 & 5 \end{array}$$

EXEMPLE : LES OPÉRATIONS ARITHMÉTIQUES

Addition de deux entiers :

$$\begin{array}{rcccc} & & 1 & & 1 \\ & 1 & 3 & 5 & 7 \\ + & 2 & 4 & 6 & 8 \\ \hline & & 8 & 2 & 5 \end{array}$$

EXEMPLE : LES OPÉRATIONS ARITHMÉTIQUES

Addition de deux entiers :

$$\begin{array}{rcccc} & & 1 & & 1 \\ & 1 & 3 & 5 & 7 \\ + & 2 & 4 & 6 & 8 \\ \hline & 3 & 8 & 2 & 5 \end{array}$$

EXEMPLE : LES OPÉRATIONS ARITHMÉTIQUES

Addition de deux entiers :

$$\begin{array}{rcccc} & & 1 & & 1 \\ & 1 & 3 & 5 & 7 \\ + & 2 & 4 & 6 & 8 \\ \hline & 3 & 8 & 2 & 5 \end{array}$$

```
def addition(nb1, nb2) :
```

```
# nb1 et nb2 entiers représentés par des tableaux de chiffres décimaux  
# (en commençant par les unités)
```

```
    res = []
```

```
    retenue = 0
```

```
# parcours parallèle des deux tableaux :
```

```
    for (chiffre1, chiffre2) in zip(nb1, nb2) :
```

```
        tmp = chiffre1 + chiffre2 + retenue
```

```
        retenue = tmp//10 # division euclidienne (en python3)
```

```
        res.append(tmp%10) # ajout à la fin du tableau
```

```
    return res + [retenue] # concaténation de 2 tableaux
```

EXEMPLE : LES OPÉRATIONS ARITHMÉTIQUES

Addition de deux entiers :

$$\begin{array}{rcccc} & & 1 & & 1 \\ & 1 & 3 & 5 & 7 \\ + & 2 & 4 & 6 & 8 \\ \hline & 3 & 8 & 2 & 5 \end{array}$$

correction : en montrant l'invariant :

« après i tours de boucle, $\text{res} = n_1 + n_2 \text{ modulo } 10^i$ »

EXEMPLE : LES OPÉRATIONS ARITHMÉTIQUES

Addition de deux entiers :

$$\begin{array}{rcccc} & & 1 & & 1 \\ & 1 & 3 & 5 & 7 \\ + & 2 & 4 & 6 & 8 \\ \hline & 3 & 8 & 2 & 5 \end{array}$$

correction : en montrant l'invariant :

« après i tours de boucle, $\text{res} = n_1 + n_2 \text{ modulo } 10^i$ »

complexité en temps : autant d'additions *élémentaires* que de chiffres dans l'écriture décimale des entiers.

\Rightarrow « complexité *linéaire* » – sous-entendu « en la *taille* ℓ des données », la taille étant ici le nombre de chiffres décimaux : dire que n_1 et n_2 sont de taille ℓ signifie que $n_1, n_2 \in O(10^\ell)$, ou encore que $\ell = 1 + \lfloor \max(\log_{10} n_1, \log_{10} n_2) \rfloor$

EXEMPLE : LES OPÉRATIONS ARITHMÉTIQUES

Multiplication de deux entiers (1)

```
def multiplication_naive(nb1, nb2) :  
    # nb1    représenté par un tableau de chiffres  
    # nb2    un entier  
    res = nb1[:]    # copie du tableau nb1  
    for i in range(2, nb2+1) : # de i=2 à i=nb2, donc nb2-1 tours  
        res = addition(res, nb1)  
    return res
```

correction : en montrant l'invariant :

« après l'étape i , $\text{res} = i \times n_1$ »

EXEMPLE : LES OPÉRATIONS ARITHMÉTIQUES

Multiplication de deux entiers (1)

```
def multiplication_naive(nb1, nb2) :  
    # nb1    représenté par un tableau de chiffres  
    # nb2    un entier  
    res = nb1[:]    # copie du tableau nb1  
    for i in range(2, nb2+1) : # de i=2 à i=nb2, donc nb2-1 tours  
        res = addition(res, nb1)  
    return res
```

correction : en montrant l'invariant :

« après l'étape i , $\text{res} = i \times n_1$ »

complexité en temps : n_2 ⁽⁻¹⁾ additions *de (grands) entiers*,
chacune étant de coût linéaire *en la taille du résultat* $n_1 n_2$ – donc
en $\log(n_1 n_2) = \log(n_1) + \log(n_2)$

\implies complexité en $O(n_2 \times \log(n_1 n_2))$,

c'est-à-dire $O(\ell \times 10^\ell)$ si les deux entiers sont de taille ℓ

EXEMPLE : LES OPÉRATIONS ARITHMÉTIQUES

Multiplication de deux entiers (2)

```
def multiplication_par_un_chiffre(nb1, chiffre2) :  
    # nb1 représenté par un tableau de chiffres  
    res = []  
    retenue = 0  
    for chiffre1 in nb1 :  
        tmp = chiffre1 * chiffre2 + retenue  
        retenue = tmp//10      # division euclidienne  
        res.append(tmp%10)  
    return res + [retenue]
```

correction : en montrant l'invariant :

« après i tours de boucle, $\text{res} \equiv n_1 \times \text{chiffre}_2 \pmod{10^i}$ »

complexité en temps : un tour de boucle par chiffre de n_1 , de coût constant

\implies complexité en $O(\log(n_1)) = O(\ell)$ si les nombres sont de taille ℓ

EXEMPLE : LES OPÉRATIONS ARITHMÉTIQUES

Multiplication de deux entiers (2)

```
def multiplication(nb1, nb2) :  
    # nb1, nb2 représentés par des tableaux de chiffres  
    res = []  
    # parcours du tableau avec itération sur les couples  
    # (indice, contenu) de chaque case  
    for (i, chiffre2) in enumerate(nb2) :  
        tmp = multiplication_par_un_chiffre(nb1, chiffre2)  
        res = addition(res, [0]*i + tmp)  
    return res
```

correction : en montrant l'invariant :

« après l'étape i , $\text{res} \equiv n_1 \times n_2 \pmod{10^i}$ »

complexité en temps : un tour de boucle par chiffre de n_2 , chacun de complexité linéaire en la taille du résultat

⇒ complexité en $O(\ell^2)$ si les nombres sont de taille ℓ

EXEMPLE : LES OPÉRATIONS ARITHMÉTIQUES

Puissance (d'un entier par exemple) (1)

```
def puissance_naive(nb1, nb2) :  
    # nb1 un élément supportant la multiplication, nb2 un entier  
    res = 1  
    for i in range(nb2) :  
        res *= nb1  
    return res
```

correction : en montrant l'invariant :

« après i tours de boucle, $res = n_1^i$ »

EXEMPLE : LES OPÉRATIONS ARITHMÉTIQUES

Puissance (d'un entier par exemple) (1)

```
def puissance_naive(nb1, nb2) :  
    # nb1 un élément supportant la multiplication, nb2 un entier  
    res = 1  
    for i in range(nb2) :  
        res *= nb1  
    return res
```

correction : en montrant l'invariant :

« après i tours de boucle, $\text{res} = n_1^i$ »

complexité : chaque tour de boucle a la complexité d'une multiplication – donc $O(\ell^2)$ par la méthode précédente pour n_1, n_2 entiers de taille ℓ

$\implies O(\ell^2 \times 10^\ell)$ si n_1, n_2 entiers de taille ℓ

EXEMPLE : LES OPÉRATIONS ARITHMÉTIQUES

Puissance (2) : *l'exponentiation binaire*

```
def puissance(nb1, nb2) :  
    # nb1 un élément supportant la multiplication, nb2 un entier  
    if nb2 == 0 : return 1  
    tmp = puissance(nb1, nb2//2)  
    carre = tmp * tmp  
    if nb2%2 == 0 : return carre  
    else : return nb1 * carre
```

correction ?

EXEMPLE : LES OPÉRATIONS ARITHMÉTIQUES

Puissance (2) : *l'exponentiation binaire*

```
def puissance(nb1, nb2) :  
    # nb1 un élément supportant la multiplication, nb2 un entier  
    if nb2 == 0 : return 1  
    tmp = puissance(nb1, nb2//2)  
    carre = tmp * tmp  
    if nb2%2 == 0 : return carre  
    else : return nb1 * carre
```

correction ? par récurrence (forte) sur n_2

EXEMPLE : LES OPÉRATIONS ARITHMÉTIQUES

Puissance (2) : *l'exponentiation binaire*

```
def puissance(nb1, nb2) :  
    # nb1 un élément supportant la multiplication, nb2 un entier  
    if nb2 == 0 : return 1  
    tmp = puissance(nb1, nb2//2)  
    carre = tmp * tmp  
    if nb2%2 == 0 : return carre  
    else : return nb1 * carre
```

correction ? par récurrence (forte) sur n_2

complexité ?

EXEMPLE : LES OPÉRATIONS ARITHMÉTIQUES

Puissance (2) : *l'exponentiation binaire*

```
def puissance(nb1, nb2) :  
    # nb1 un élément supportant la multiplication, nb2 un entier  
    if nb2 == 0 : return 1  
    tmp = puissance(nb1, nb2//2)  
    carre = tmp * tmp  
    if nb2%2 == 0 : return carre  
    else : return nb1 * carre
```

correction ? par récurrence (forte) sur n_2

complexité ? chaque appel récursif nécessite 1 ou 2 multiplications, et le nombre d'appels est égal à $\lfloor \log_b n_2 \rfloor + 1$,
donc $O(\log \ell)$ *multiplications* si n_2 est un entier de taille ℓ