Module EA4 – Éléments d'Algorithmique II

Dominique Poulalhon
dominique.poulalhon@liafa.univ-paris-diderot.fr

Université Paris Diderot
L2 Informatique
Année universitaire 2014-2015

Organisation du module

Emploi du temps

- Cours : 2h par semaine, mercredi 10h30-12h30, amphi 13E
- TD : 2h par semaine
  - Info1 : mardi 14h30 – 477F – Rémy Chrétien
    remy_chretien@yahoo.fr
  - Info2 : vendredi 8h30 – 515B – Enrica Duchi
    enrica.duchi@liafa.univ-paris-diderot.fr
  - Info3 : jeudi 10h30 – 476F – Mehdi Bouaziz
    mehdi.bouaziz@ens.fr
  - Info4 : jeudi 14h30 – 2707F – Nathanael François
    nathanael.francois@liafa.univ-paris-diderot.fr
- TP : 2h une semaine sur deux
- semaine 1 des TD et TP : mardi 27 janvier

Emploi du temps

- Cours : 2h par semaine, mercredi 10h30-12h30, amphi 13E
- TD : 2h par semaine
- TP : 2h une semaine sur deux
  - Info1 et Info4 : mardi 16h30 – 531C
  - Info2 et Info3 : mercredi 8h30 – 532C
  - groupes impairs en semaine impaire
  - groupes pairs en semaine paire
- semaine 1 des TD et TP : mardi 27 janvier

Organisation du module

Emploi du temps

- Cours : 2h par semaine, mercredi 10h30-12h30, amphi 13E
- TD : 2h par semaine
- TP : 2h une semaine sur deux
- semaine 1 des TD et TP : mardi 27 janvier

Pour nous écrire, toujours mentionner `[EA4]` dans le sujet

Un site DiDEL pour toutes les annonces (et les sujets...)

donc : Inscrivez-vous !

MCC

- 60% examen final
- 40% contrôle continu (2 ou 3 interros, sur papier ou machine)

Éléments d'« Algorithmique » ?

Éléments d'« Algorithmique » ?

= « conception et étude des algorithmes »

Éléments d'« Algorithmique » ?

= « conception et étude des algorithmes »

« algorithme » ?

Éléments d'« Algorithmique » ?

$=$ « conception et étude des algorithmes »

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

Éléments d'« Algorithmique » ?

= « conception et étude des algorithmes »

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

pas limité au domaine informatique – d'ailleurs, de nombreux algorithmes ont été décrits bien avant l'invention des ordinateurs :

- des algorithmes de calcul
- des constructions géométriques
- des recettes de cuisine
- des manuels de construction...

ÉLÉMENTS D'« ALGORITHMIQUE » ?

= « conception et étude des algorithmes »

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

pas limité au domaine informatique – d'ailleurs, de nombreux algorithmes ont été décrits bien avant l'invention des ordinateurs :

- des algorithmes de calcul
- des constructions géométriques
- des recettes de cuisine
- des manuels de construction...

Exemple : comment nourrir un loup ami des lapins ?
(©Jean-Luc Fromental, Grégoire Solotareff)

Éléments d'« Algorithmique » ?

= « conception et étude des algorithmes »

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

pas limité au domaine informatique – d'ailleurs, de nombreux algorithmes ont été décrits bien avant l'invention des ordinateurs :

- des algorithmes de calcul
- des constructions géométriques
- 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

Éléments d'« Algorithmique » ?

= « conception et étude des algorithmes »

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

Éléments d'« Algorithmique » ?

= « conception et étude des algorithmes »

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

trois axes d'étude :

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

Éléments d'« Algorithmique » ?

= « conception et étude 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

Éléments d'« Algorithmique » ?

= « conception et étude 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 ?

Éléments d'« Algorithmique » ?

= « conception et étude 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 ?

Éléments d'« Algorithmique » ?

= « conception et étude 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 :

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

Exemple : les opérations arithmétiques

Addition de deux entiers :

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

Exemple : les opérations arithmétiques

Addition de deux entiers :

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

Exemple : les opérations arithmétiques

Addition de deux entiers :

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

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 :

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

```
def addition(nb1, nb2) :
# entiers représentés par des tableaux de chiffres décimaux
  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 :

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

correction : en montrant l'invariant :

« après l'étape i, $\mathtt{res} = n_1 + n_2$ *modulo* 10^i »

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

$\Longrightarrow$ « complexité linéaire » (en la taille ℓ des données, la taille étant ici le nombre de chiffres décimaux : $n_1, n_2 \in O(10^\ell)$)

Exemple : les opérations arithmétiques

Multiplication de deux entiers (1)

```
def multiplication(nb1, nb2) :
# nb1, nb2 toujours dans des tableaux
# valeur(nb) retourne l'entier correspondant
  res = nb1[:]
  for i in range(1, valeur(nb2)) : # répéter nb2-1 fois
    res = addition(res, nb1)
  return res
```

correction : en montrant l'invariant :

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

complexité en temps : $n_2(-1)$ additions de (grands) entiers, chacune étant de coût linéaire en la taille de n_1

$\Longrightarrow$ complexité en $O(n_2 \times \log(n_1)) = O(\ell \times 10^{\ell})$

Exemple : les opérations arithmétiques

Multiplication de deux entiers (2)

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

correction ?

complexité ?

Exemple : les opérations arithmétiques

Multiplication de deux entiers (2)

```
def multiplication(nb1, nb2) :
  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 ?

complexité ?

Exemple : les opérations arithmétiques

Multiplication de deux entiers (3) : la méthode du paysan russe

```
def multiplication_russe(nb1, nb2) :
# cette fois, les entiers sont vraiment des entiers
  res = 0
  while nb2 != 0 :
    if nb2%2 == 1 : res += nb1
    nb1 *= 2        # ou : nb1 << 1
    nb2 //= 2       # ou : nb2 >> 1
  return res
```

correction ?

complexité ?

Exemple : les opérations arithmétiques

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

```
def puissance(nb1, nb2) :
  res = 1
  for i in range(nb2) :
    res *= nb1
  return res
```

correction ?

complexité ?

Exemple : les opérations arithmétiques

Puissance (2) : l'exponentiation binaire

```
def puissance(nb1, nb2) :
  if nb2 == 0 : return 1
  tmp = puissance(nb1, nb2//2)
  carre = tmp * tmp
  if nb2%2 == 0 : return carre
  else : return nb1 * carre
```

correction ?

complexité ?

Application : calcul du n^e terme de la suite de Fibonacci

suite définie par $F_0 = 0$, $F_1 = 1$ et

$$\forall n \geqslant 2, \; F_n = F_{n-1} + F_{n-2}$$

Application : calcul du n^e terme de la suite de Fibonacci

suite définie par $F_0 = 0$, $F_1 = 1$ et

$$\forall n \geqslant 2,\ F_n = F_{n-1} + F_{n-2}$$

- utiliser directement la définition par récurrence

```
def fibo(n) :
  if n <= 0 : return 0
  if n <= 2 : return 1
  return fibo(n-1) + fibo(n-2)
```

Application : calcul du n^e terme de la suite de Fibonacci

suite définie par $F_0 = 0$, $F_1 = 1$ et

$$\forall n \geqslant 2,\ F_n = F_{n-1} + F_{n-2}$$

- utiliser directement la définition par récurrence
- garder un tableau de toutes les premières valeurs

```
def fibo(n) :
  if n <= 0 : return 0
  liste = [0, 1]
  for i in range(1, n) :
    liste.append(liste[i-1] + liste[i])
  return liste[n]
```

Application : calcul du n^e terme de la suite de Fibonacci

suite définie par $F_0 = 0$, $F_1 = 1$ et

$$\forall n \geqslant 2,\ F_n = F_{n-1} + F_{n-2}$$

- utiliser directement la définition par récurrence
- garder un tableau de toutes les premières valeurs
- garder seulement les deux dernières valeurs

```
def fibo(n) :
  if n <= 0 : return 0
  previous, last = 0, 1
  for i in range(1, n) :
    previous, last = last, previous + last
  return last
```

Application : calcul du n^e terme de la suite de Fibonacci

suite définie par $F_0 = 0$, $F_1 = 1$ et

$$\forall n \geqslant 2,\ F_n = F_{n-1} + F_{n-2}$$

- utiliser :

$$\forall n \geqslant 1,\ \begin{bmatrix} F_{n+1} \\ F_n \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

  et donc :

$$\forall n \geqslant 1,\ \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix}$$

```
def fibo(n) :
  M = [ [1, 1], [1, 0] ]
  M = puissance_matrice_2_2 (M, n-1)
  return M[0][0]
```

Exemple : les opérations arithmétiques

Multiplication de deux entiers (4) : la méthode de Karatsuba

(sur des polynômes pour éviter les retenues)

Hypothèse : P, Q de degré (au plus) $2^k - 1$

$P^{(0)}$ et $P^{(1)}$ les polynômes de degré (au plus) $2^{k-1} - 1$ tels que :

$$P = P^{(0)} + P^{(1)} \cdot X^{2^{k-1}}$$

Alors :

$$P \cdot Q = P^{(0)}Q^{(0)} + (P^{(0)}Q^{(1)} + P^{(1)}Q^{(0)})X^{2^{k-1}} + P^{(1)}Q^{(1)}X^{2^k}$$

Ou encore :

$$\begin{aligned} P \cdot Q = {} & P^{(0)}Q^{(0)} + P^{(1)}Q^{(1)}X^{2^k} \\ & + \left[(P^{(0)} + P^{(1)})(Q^{(0)} + Q^{(1)}) - P^{(0)}Q^{(0)} - P^{(1)}Q^{(1)}\right] X^{2^{k-1}} \end{aligned}$$