U.E. Éléments d'Algorithmique 1 (EA3)

Balthazar Bauer <balthazarbauer@aol.com>
Christine Tasson <tasson@irif.fr>

Université Paris Diderot

Semestre d'automne 2020

Organisation de l'UE

- Responsable de l'UE : Enrica Duchi <duchi@irif.fr>
- Enseignants : C. Tasson (18/09, 2/10 et 9/10), B. Bauer (25/09 et à paritir du 16/10)
- 12 séances de Cours-TD de 2h30 heures chacunes
- Évaluations : 4 contrôles de 45 minutes à 1heure en début du cours-td :
  - CC1 : semaine du 28 septembre => Vendredi 2 octobre
  - CC2 : semaine du 12 octobre => Vendredi 16 octobre
  - CC3 : semaine du 9 novembre
  - CC4 : semaine du 30 novembre

  Note session 1 = Moyenne de tous les CC
  Note session 2 = max(ET, MCC) où MCC = moyenne des CC où on remplace le dernier CC par l'ET, ET = examen session2.
- Pas de cours la semaine du 26 octobre.

Moodle = plateforme d'enseignements

Se connecter

https://moodle.u-paris.fr/

- S'inscrire : L2 Licence informatique / IF13Y020 Eléments D'algorithmique 1
- S'inscrire dans son groupe avec la clef : `EA3groupe:5` ou `EA3groupe:6`
- Rendre un exercice (chaque semaine)

But de l'UE

- Algorithmique (conception, preuve, complexité)
- Programmation

Il faut programmer : un des exercices de TD chaque semaine à déposer sur Moodle après l'avoir exécuté et **testé** :

- Installer Java et compiler sur sa machine
  - `javac Test.java`
  - `java Test`
- Programmer en ligne :
  - java_visualize
  - java_tutor

Algorithme ou programme qu'elle différence ? - Gérard Berry

Definition

Un *algorithme* est

Definition

Un *programme* logiciel est

Algorithme ou programme qu'elle différence ? - Gérard Berry

Definition

Un *algorithme* est un objet abstrait définissant un calcul, exprimé en langage mathématique, et analysable mathématiquement.

Definition

Un *programme* logiciel est

Algorithme ou programme qu'elle différence ? - Gérard Berry

Definition

Un *algorithme* est un objet abstrait définissant un calcul, exprimé en langage mathématique, et analysable mathématiquement.

Definition

Un *programme* logiciel est un objet éminemment concret, (...) écrit dans un langage de programmation, qui cherche à rendre plus humain les langages cryptiques des machines.

Algorithme ou programme qu'elle différence ? - Gérard Berry

Definition

Un *algorithme* est un objet abstrait définissant un calcul, exprimé en langage mathématique, et analysable mathématiquement.

Definition

Un *programme* logiciel est un objet éminemment concret, (...) écrit dans un langage de programmation, qui cherche à rendre plus humain les langages cryptiques des machines.

Ce sont les programmes qui sont exécutés par les ordinateurs. Un programme reste à un niveau de détail très supérieur à celui d'un algorithme

À l'origine, un problème à résoudre

Problème : Faire un gâteau au chocolat. • **Données en entrée :** les ingrédients du gâteau. • **Algorithme :** la recette du gâteau. • **Sortie :** le gâteau au chocolat !	**Problème : Trouver le minimum entre deux entiers**.

Programmer en `java` l'algorithme décrit en pseudocode ci-dessus, et le tester ; L'exécuter sur machine ; L'exécuter sur java_visualize

À l'origine, un problème à résoudre

Problème : Faire un gâteau au chocolat.

- **Données en entrée :** les ingrédients du gâteau.
- **Algorithme :** la recette du gâteau.
- **Sortie :** le gâteau au chocolat !

Problème : Trouver le minimum entre deux entiers.

- **Données en entrée :** 2 entiers **i** et **j**.

Programmer en `java` l'algorithme décrit en pseudocode ci-dessus, et le tester ; L'exécuter sur machine ; L'exécuter sur java_visualize

À l'origine, un problème à résoudre

Problème : Faire un gâteau au chocolat.

- **Données en entrée :** les ingrédients du gâteau.
- **Algorithme :** la recette du gâteau.
- **Sortie :** le gâteau au chocolat !

Problème : Trouver le minimum entre deux entiers.

- **Données en entrée :** 2 entiers **i** et **j**.
- **Algorithme :** (pseudocode)

```
si (i>j) alors
      min=j
sinon
      min=i
```

Programmer en `java` l'algorithme décrit en pseudocode ci-dessus, et le tester ; L'exécuter sur machine ; L'exécuter sur java_visualize

À l'origine, un problème à résoudre

Problème : Faire un gâteau au chocolat.

- **Données en entrée :** les ingrédients du gâteau.
- **Algorithme :** la recette du gâteau.
- **Sortie :** le gâteau au chocolat !

Problème : Trouver le minimum entre deux entiers.

- **Données en entrée :** 2 entiers **i** et **j**.
- **Algorithme :** (pseudocode)

```
si (i>j) alors
        min=j
sinon
        min=i
```

- **Sortie :** la valeur min.

Programmer en `java` l'algorithme décrit en pseudocode ci-dessus, et le tester ; L'exécuter sur machine ; L'exécuter sur java_visualize

Analyse d'un algorithme : Terminaison et Correction

Definition

Une **instance** d'un problème est un ensemble de données sur lesquelles nous allons exécuter notre algorithme.

Par exemple

```
(i=1, j=-3)
```

est une instance du problème de recherche d'un minimum entre deux entiers.

Analyse d'un algorithme : Terminaison et Correction

Definition

Une **instance** d'un problème est un ensemble de données sur lesquelles nous allons exécuter notre algorithme.

Par exemple

```
(i=1, j=-3)
```

est une instance du problème de recherche d'un minimum entre deux entiers.

Definition

Un algorithme **termine** si, pour chaque instance du problème (entrées), il renvoie une sortie en un nombre finie d'étapes.

Analyse d'un algorithme : Terminaison et Correction

Definition

Une **instance** d'un problème est un ensemble de données sur lesquelles nous allons exécuter notre algorithme.

Par exemple

```
(i=1, j=-3)
```

est une instance du problème de recherche d'un minimum entre deux entiers.

Definition

Un algorithme **termine** si, pour chaque instance du problème (entrées), il renvoie une sortie en un nombre finie d'étapes.

Definition

Un algorithme est **correct** s'il donne la bonne solution (la solution attendue) pour chaque instance du problème.

Analyse d'un algorithme : Complexité

- **Complexité en temps de l'algorithme**
  - compter le nombre d'opérations élémentaires effectuées lors de l'exécution de l'algorithme
  - pour avoir une bonne complexité il faut bien choisir les structures de données utilisées

Nous allons nous concentrer sur la complexité en temps de l'algorithme.

Analyse d'un algorithme : Complexité

- **Complexité en temps de l'algorithme**
  - compter le nombre d'opérations élémentaires effectuées lors de l'exécution de l'algorithme
  - pour avoir une bonne complexité il faut bien choisir les structures de données utilisées
- **Complexité en espace de l'algorithme**
  - mesurer la place mémoire maximale occupée durant l'exécution
  - là encore, il faut bien choisir les structures de données utilisées

Nous allons nous concentrer sur la complexité en temps de l'algorithme.

Recherche d'un élément dans un tableau

Algorithme en pseudocode

```
Données: tab: un tableau de n entiers,
         el: élément
Pseudocode:
  for i from 0 to n-1
    if el == tab[i] then return i
  return -1
```

Quelles sont les opérations élémentaires ?
Quelle est la complexité en temps de cet algorithme ?

Recherche d'un élément dans un tableau

Algorithme en pseudocode

```
Données: tab: un tableau de n entiers,
         el: élément
Pseudocode:
  for i from 0 to n-1
    if el == tab[i] then return i
  return -1
```

Quelles sont les opérations élémentaires ? comparaison : `el==tab[i]`
Quelle est la complexité en temps de cet algorithme ?

Recherche d'un élément dans un tableau

Algorithme en pseudocode

```
Données: tab: un tableau de n entiers,
         el: élément
Pseudocode:
  for i from 0 to n-1
    if el == tab[i] then return i
  return -1
```

Quelles sont les opérations élémentaires ? comparaison : `el==tab[i]`
Quelle est la complexité en temps de cet algorithme ?
Dans le cas où l'élément recherché est à la fin du tableau, nous allons faire n opérations élémentaires, donc la complexité est linéaire dans le pire cas.

Solution efficace d'un problème ?

Opérations élémentaires

Solution efficace d'un problème ?

Opérations élémentaires

- affectations : `i=5`
- comparaisons : `i>j`
- accès à un élément d'un tableau : `tab[i]`

Ces trois opérations élémentaires se font en temps constant (ne dépend pas de la taille des données).

Solution efficace d'un problème ?

Opérations élémentaires

- affectations : `i=5`
- comparaisons : `i>j`
- accès à un élément d'un tableau : `tab[i]`

Ces trois opérations élémentaires se font en temps constant (ne dépend pas de la taille des données).

Definition

La **complexité** d'un algorithme est le nombre d'**opération élémentaire** pour retourner la sortie.

On étudie la complexité dans le meilleur des cas, dans tous les cas, dans le pire des cas, ou encore en moyenne.

Ordre de grandeurs

Hypothèse théorique : 1 opération élémentaire par cycle de processeur 10ns

n	20	40	60	100	300
n^2	$\frac{1}{2500}$ ms	$\frac{1}{625}$ ms	$\frac{1}{278}$ ms	$\frac{1}{100}$ ms	$\frac{1}{11}$ ms
n^5	3 ms	$\frac{1}{10}$ s	$\frac{78}{100}$ s	10 s	$40,5$ min
2^n	1 ms	$18,3$ min	$36,5$ années	4.10^{11} siècles	...
n^n	$3,3.10^7$ siècles	...			

← taille de la donnée

↑ nombres d'opérations élémentaires

Ordre de grandeurs

Hypothèse théorique : 1 opération élémentaire par cycle de processeur 10ns

n	20	40	60	100	300
n^2	$\frac{1}{2500}$ ms	$\frac{1}{625}$ ms	$\frac{1}{278}$ ms	$\frac{1}{100}$ ms	$\frac{1}{11}$ ms
n^5	3 ms	$\frac{1}{10}$ s	$\frac{78}{100}$ s	10 s	$40,5$ min
2^n	1 ms	$18,3$ min	$36,5$ années	4.10^{11} siècles	...
n^n	$3,3.10^7$ siècles	...			

taille de la donnée

nombres d'opérations élémentaires

La **complexité** en temps permet de comprendre si un algorithme peut servir pour de grandes taille de données.

Ordre de grandeurs

Hypothèse théorique : 1 opération élémentaire par cycle de processeur 10ns

n	20	40	60	100	300
n^2	$\frac{1}{2500}$ ms	$\frac{1}{625}$ ms	$\frac{1}{278}$ ms	$\frac{1}{100}$ ms	$\frac{1}{11}$ ms
n^5	3 ms	$\frac{1}{10}$ s	$\frac{78}{100}$ s	10 s	$40,5$ min
2^n	1 ms	$18,3$ min	$36,5$ années	4.10^{11} siècles	...
n^n	$3,3.10^7$ siècles	...			

← taille de la donnée

↑ nombres d'opérations élémentaires

La **complexité** en temps permet de comprendre si un algorithme peut servir pour de grandes taille de données.

Attention : en pratique ça peut être différent du temps **mesuré** pour exécuter un programme implémentant un algorithme.

Problèmes difficiles, voire infaisables

- Il existe des algorithmes pour lesquels nous ne connaissons pas de solutions efficaces. Exemple : le problème du voyageur de commerce. (voir l'excellent Article d'Interstice avec une application qui permet de mannipuler ce problème classique de l'algorithmique.

Problèmes difficiles, voire infaisables

- Il existe des algorithmes pour lesquels nous ne connaissons pas de solutions efficaces. Exemple : le problème du voyageur de commerce. (voir l'excellent Article d'Interstice avec une application qui permet de mannipuler ce problème classique de l'algorithmique.
- Il existe des problèmes qu'on ne sait pas résoudre avec un algorithme. Exemple : Le problème de l'arrêt : peut-on écrire un algorithme A qui pour tout algorithme B et toute instance I détermine si B s'arrête pour la donnée I ?

Problèmes difficiles, voire infaisables

- Il existe des algorithmes pour lesquels nous ne connaissons pas de solutions efficaces. Exemple : le problème du voyageur de commerce. (voir l'excellent Article d'Interstice avec une application qui permet de mannipuler ce problème classique de l'algorithmique.
- Il existe des problèmes qu'on ne sait pas résoudre avec un algorithme. Exemple : Le problème de l'arrêt : peut-on écrire un algorithme A qui pour tout algorithme B et toute instance I détermine si B s'arrête pour la donnée I ?
  La réponse est non, c'est un problème **indécidable**.
  Une vidéo qui présente pourquoi le problème de l'arrêt ne peut pas être résolu.