Éléments d'algorithmique Cours-TD 1

L2 Informatique

Université de Paris

15-09-2020

Modalités

$\Rightarrow$ email : combe@irif.fr (Camille Combe)

$\Rightarrow$ 4 contrôles de 45 minutes-1 heure en début du cours-td.

- CC1 : semaine du 28 septembre,
- CC2 : semaine du 12 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 session 2.

$\Rightarrow$ BONUS : Coder un des exercices de TD chaque semaine et le déposer sur Moodle.
`http://www.pythontutor.com/java.html`

Traitement de l'information

Les ordinateurs sont utilisés pour

- le traitement d'informations, par exemple un GPS,
- le stockage d'informations, par exemple stockage de films ou de pdf.

Réception d'informations $\rightarrow$ Traitement des informations $\rightarrow$ Émission d'informations déduites

Définition d'un programme

Tout traitement demandé à la machine, par l'utilisateur, est effectué par l'exécution séquencée d'opérations appelées instructions. Une suite d'instructions est appelée un programme.

Définition

Un programme est une suite d'instructions permettant à un système informatique d'exécuter une tâche donnée.

La tâche donnée doit être écrite dans un langage de programmation compréhensible par un ordinateur (Java, Python, C, etc.).

Définition

Un langage de programmation est un ensemble de règles de vocabulaire et de grammaire compréhensible par un ordinateur.

Données $\rightarrow$ Programme $\rightarrow$ Résultats

Algorithme

Un algorithme est une suite finie d'instructions qui prend des données en entrée et donne un résultat en sortie en temps fini. À l'origine il y a 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:

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

- Sortie: la valeur min.

Algorithme correct

Une instance d'un problème est un jeu 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.

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

Pour récapituler

Définition

Un algorithme est

- une description formelle d'un procédé de traitement qui permet, à partir d'un ensemble d'informations initiales, d'obtenir des informations déduites,
- une succession finie et non ambigüe d'opérations,

et son exécution se termine toujours.

Définition

Un programme

- est une suite d'instructions définies dans un langage donné,
- décrit un algorithme.

Efficacité d'un algorithme

- **Complexité en temps**
  - compter le nombre d'opérations élémentaires effectuées lors de l'exécution de l'algorithme,
  - bien choisir les structures de données utilisées.
- **Complexité en espace**
  - mesurer la place mémoire maximale occupée durant l'exécution,
  - bien choisir les structures de données utilisées.
- **Terminaison :** termine en un temps fini.
- **Complétude :** pour un espace de problèmes donné l'algorithme donne toujours des propositions de solutions.

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

Temps de calcul

Sur les machines actuelles, le temps pris par un calcul est **très difficile à prévoir**, avec une forte composante aléatoire :

- ⋆ traduction (interprétation, compilation),
- ⋆ forte dépendance à l'environnement (mémoire, système d'exploitation, multi-threading,...)
- ⋆ nombreuses optimisations qui dépendent de l'historique (caches, ...).

$\Longrightarrow$ Nous travaillons avec un modèle de machine simplifiée.

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

Complexités

Définition

- ⋆ Complexité temporelle : (ou en temps) temps de calcul,
- ⋆ Complexité spatiale : (ou en espace) l'espace mémoire requis par le calcul.

Définition

- ⋆ La complexité pratique est une mesure précise des complexités temporelles et spatiales pour un modèle de machine donné.
- ⋆ La complexité (théorique) est un ordre de grandeur de ces couts, exprimé de manière la plus indépendante possible des conditions pratiques d'exécution.

Analyse de la complexité en temps

pseudocode	java

`RechercheMin (tab: un tableau de n entiers){`
`  min <- tab[0]`
`  for i from 1 to n-1 {`
`  if min > tab[i] {`
`  min <- tab[i]`
`  }`
`  }`
`  renvoyer min`
`}`
	`public static int min (int [] tab){`
	`  int min=tab[0];`
	`  for (int i=1; i<=tab.length-1; i++){`
	`  if (min > tab[i]) {`
	`  min=tab[i];`
	`  }`
	`  }`
	`  return min;`
	`}`

Opérations élémentaires :

- ⋆ assignations: `min=tab[0]`, `min=tab[i]` ,
- ⋆ comparaisons: `min > tab[i]`,
- ⋆ 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).

Analyse de la complexité en temps

Complexité en temps de l'algorithme :
Nous allons faire au pire $4(n-1)+2$ opérations élémentaires, où n est la longueur du tableau, donc la complexité est linéaire. Nous remarquons que dans le cas de recherche d'un minimum nous ne pouvons pas faire mieux, puisque nous sommes bien obligés de parcourir tout le tableau.

Recherche d'un élément dans un tableau

pseudocode	java

`RechercheElem (tab: un tableau de n entiers, el: élément){`
`  for i from 0 to n-1 {`
`  if el == tab[i] {`
`  return i`
`  }`
`  }`
`  return -1`
`}`
	`public static int rec (int [] tab, int el){`
	`  for (int i=0; i<=tab.length-1; i++) {`
	`  if (el==tab[i]) {`
	`  return i;`
	`  }`
	`  }`
	`  return -1;`
	`}`

Opérations élémentaires :

- comparaison: `el==tab[i]`

Complexité en temps de l'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.

Pour terminer

- Un algorithme est efficace si sa complexité est au plus un polynôme en n, où n est la taille des données en entrée.
- Il existe des algorithmes pour lesquels nous ne connaissons pas de solutions efficaces. Exemple: le problème du voyageur de commerce.
- 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 semi-algorithme B et toute instance I détermine si B s'arrête pour la donnée I ? La réponse est non, c'est un problème indécidable.