Elements d’Algorithmique
CMTD1: Introduction

Université de Paris Cité, IRIF

Organisation de l'UE

- Responsable de l'UE : Enrica Duchi duchi@irif.fr
- 12 séances de Cours-TD de 2h30 chacune. Semaine de pause: 30 octobre.
- MCC : 2 cc en cours de 1h, 1 contrôle en amphi la semaine du 18 décembre
  - Note session1=$\max\left(\frac{CC1}{4}+\frac{CC2}{4}+\frac{CC3}{2}, CC3\right)$

Absence au CC3: session2

Absence injustifiée CC1 ou CC2: session2

Absence justifiée aux CC1 ou CC2:
la note sera calculée sur les autres CC

  - Note session2=$\max\left(\frac{CC1}{4}+\frac{CC2}{4}+\frac{E}{2}, E\right)$ ou Note session2=E si AI aux CC1 ou CC2

    où E est l'examen de la session 2.

Contrôles et devoirs maison

- CC1 : semaine du 9 octobre
- CC2 : semaine du 13 novembre
- CC3 : contrôle en amphi, semaine du 18 décembre
- devoir maison à faire chaque semaine : exercice à implementer en java.

Dans le CC3 il y aura une ou deux questions sur les DM que vous avez travaillés pendant l'année: réponse en java!

Moodle, plateforme d’enseignements

- Se connecter: https://moodle.u-paris.fr
- S’inscrire : IF13Y020 Eléments d’algorithmique 1 dans le bon groupe!
- Vous y trouverez cours et sujet de TD chaque semaine.
- Toute autre information concernant le cours

But de l’UE

- Algorithmique (conception, preuve, complexité)
- Programmation:
  - Installer java sur sa machine.
  - Programmer en ligne https://pythontutor.com/java.html

Qu’est-ce qu’un algorithme ?

Un algorithme est une suite finie d’instructions qui prend des données en entrée et donne un résultat en sortie.

À l’origine il y a un problème à résoudre. Par exemple :

Problème : Faire un gâteau au chocolat.

Problème : Trouver le minimum entre deux entiers.

Qu’est-ce qu’un algorithme ?

Un algorithme est une suite finie d’instructions qui prend des données en entrée et donne un résultat en sortie.

À l’origine il y a un problème à résoudre. Par exemple :

Problème : Faire un gâteau au chocolat.

- **Données en entrée :** 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**,**j**.
- **Algorithme :**
  1: **si**(i>j) **alors** min=j
  2: **sinon** min=i
- **Sortie :** la valeur min.

Qu’est-ce qu’un algorithme ?

Problème : Trouver le minimum entre deux entiers.

Qu’est-ce qu’un algorithme ?

Problème : Trouver le minimum entre deux entiers.

- **Données en entrée :**2 entiers **i**, **j**.
- **Algorithme :**

  1: **si**(i>j) **alors** min=j

  2: **sinon** min=i
- **Sortie :** la valeur min.
- Une **instance** d’un problème est un ensemble de données sur les quelles nous allons exécuter notre algorithme.

  Par exemple i=1, j=3 est une instance du problème de recherche d’un minimum entre deux entiers.

Qu’est-ce qu’un algorithme ?

Problème : Trouver le minimum entre deux entiers.

- **Données en entrée :**2 entiers **i**, **j**.
- **Algorithme :**

  1: **si**(i>j) **alors** min=j

  2: **sinon** min=i
- **Sortie :** la valeur min.
- Une **instance** d’un problème est un ensemble de données sur les quelles 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 pour chaque instance d’un problème.

Écriture d’un algorithme

- Écriture en pseudo-code. À utiliser en cours ou en CC.
- Écriture en java. Implementer les devoirs maisons en java. Une ou deux questions à repondre en java au CC3.

Efficacité d’un algorithme

- **Complexité en temps de l’algorithme**

Efficacité d’un algorithme

- **Complexité en temps de l’algorithme**
  - compter le nombre d’opérations élémentaires effectuées lors de l’exécution de l’algorithme

Efficacité d’un algorithme

- **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

Efficacité d’un algorithme

- **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**

Efficacité d’un algorithme

- **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

Efficacité d’un algorithme

- **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

Efficacité d’un algorithme

- **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.

Analyse de la complexité en temps pour la recherche d'un mininum dans un tableau

pseudocode	java

Entrée : tableau de n entiers tab
1: **fonction** RechercheMin(tab)
2: min← tab[0]
3: **pour** i← 1à n– 1 **faire**
4: **si** min>tab[i] **alors**
5: min← tab[i]
retourne 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;
	}

Quelles sont les opérations élémentaires ?

Analyse de la complexité en temps pour la recherche d'un mininum dans un tableau

pseudocode	java

Entrée : tableau de n entiers tab
1: **fonction** RechercheMin(tab)
2: min← tab[0]
3: **pour** i← 1 à n– 1 **faire**
4: **si** min>tab[i] **alors**
5: min← tab[i]
retourne 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;
	}

Quelles sont les opérations élémentaires ?

- assignations : min=tab[0],min=tab[i]
- comparaisons : min>tab[i]
- L'accès à un élément tab[i] d'un tableau se fait en temps constant.

Recherche d’un élément dans un tableau

pseudocode	java

Entrée :tableau de n entiers tab
Entrée :un entiers el
1: **fonction** RechercheElem(tab)
2: **pour** i←0 à n−1 **faire**
3: **si** el==tab[i] **alors retourne** i
retourne -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;
	}

Quelles sont les opérations élémentaires ?

- comparaisons :el==tab[i]

Quelle est la complexité entemps 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.

Pourquoi on s'intéresse à la complexité en temps ?

La complexité en temps permet de comprendre si un algorithme peut servir pour de grandes tailles de données :

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

Pour terminer

- Il existe des algorithmes pour lesquels nous ne connaissons pas de solutions efficaces.

  Le problème du voyageur de commerce:

  Etant donné un ensemble de villes, determiner le plus court circuit passant par chaque ville une seule fois.

- Il existe des problèmes qu'on ne sait pas résoudre avec un algorithme.

  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**.