Elements d'Algorithmique
CMTD1: Introduction

Université de Paris, IRIF

Organisation de l'UE

Organisation de l'UE

- Responsable de l'UE : Enrica Duchi `duchi@irif.fr`
- 12 séances de Cours-TD de 2h30 chacune. Une pause la semaine du 1er novembre.
- Modalités de contrôle des connaissances : 3 contrôles en cours de 1h en début du cours-td, 2 devoirs maison notés. Un rattrappage pour la session 2.
  - Note DM$=\frac{DM1 + DM2}{2}$
  - Note session1$=\frac{CC1 + CC2 + CC3 + DM}{4}$
  - Note session2=max(E, $\frac{CC1 + CC2 + E + DM}{4}$), où E est le rattrappage en session 2.

Contrôles et devoirs maison

Contrôles et devoirs maison

- CC1 : semaine du 11 octobre
- CC2 : semaine du 15 novembre
- CC2 : semaine du 13 décembre
- devoir maison à deposer sur Moodle chaque semaine : deux DM pris au hasard sont notés (un sur les premières 6 séances, l'autre sur les 6 dernières).
  Il est donc important de rendre les devoirs maison ! Vous avez une semaine pour les déposer sur Moodle.

Moodle, plateforme d'enseignements

Moodle, plateforme d'enseignements

- Se connecter : `https://moodle.u-paris.fr`
- S'inscrire : IF13Y020 Eléments d'algorithmique 1.
- S'inscrire dans son propre groupe avec la clef `EA3groupe:i`, où `i` est le numéro de votre groupe.
- Deposer un DM sur moodle chaque semaine dans le depôt de votre groupe.

But de l'UE

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 :** 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 :**
  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** et **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 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.

Qu'est-ce qu'un *algorithme* ?

Problème : Trouver le minimum entre deux entiers.

- **Données en entrée :** 2 entiers **i** et **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 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 pour chaque instance d'un problème.

Écriture d'un algorithme

Écriture d'un algorithme

- Écriture en pseudo-code. À utiliser en cours ou en CC.
- Écriture en java. À utiliser pour les devoirs maisons à rendre chaque semaine mais aussi en cours ou en CC pour qui le souhaite.

Efficacité d'un algorithme

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:     n ← tab[0]
3:     pour i ← 0 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 ?

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:     n ← tab[0]
3:     pour i ← 0 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** $i \leftarrow 0$ 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 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.

Pourquoi on s'intéresse à la complexité en temps ?

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

Pour terminer

- 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 algorithme B et toute instance I détermine si B s'arrête pour la donnée I ?
  La réponse est non, c'est un problème **indécidable**.