

Algorithmique

Programmation Objet

Python



Andrea G. B. Tettamanzi

Université de Nice Sophia Antipolis

Département Informatique

andrea.tettamanzi@unice.fr

CM - Séance 2

Introduction à l'analyse des algorithmes

Plan

- Complexité des problèmes et coût des algorithmes
- Notations O , o , Θ et Ω
- Classes de complexité
- Pseudo-langage

Complexité

- Tous les algorithmes ne sont pas équivalents.
- On les différencie selon au moins 2 critères :
 - Temps de calcul : lents vs rapides
 - Mémoire utilisée : peu vs beaucoup
- On parle de complexité
 - en temps (vitesse)
 - en espace (mémoire utilisée)
- Indicateur de l'efficacité d'un algorithme et de la difficulté du problème
- Analyse des algorithmes = étude de leur complexité

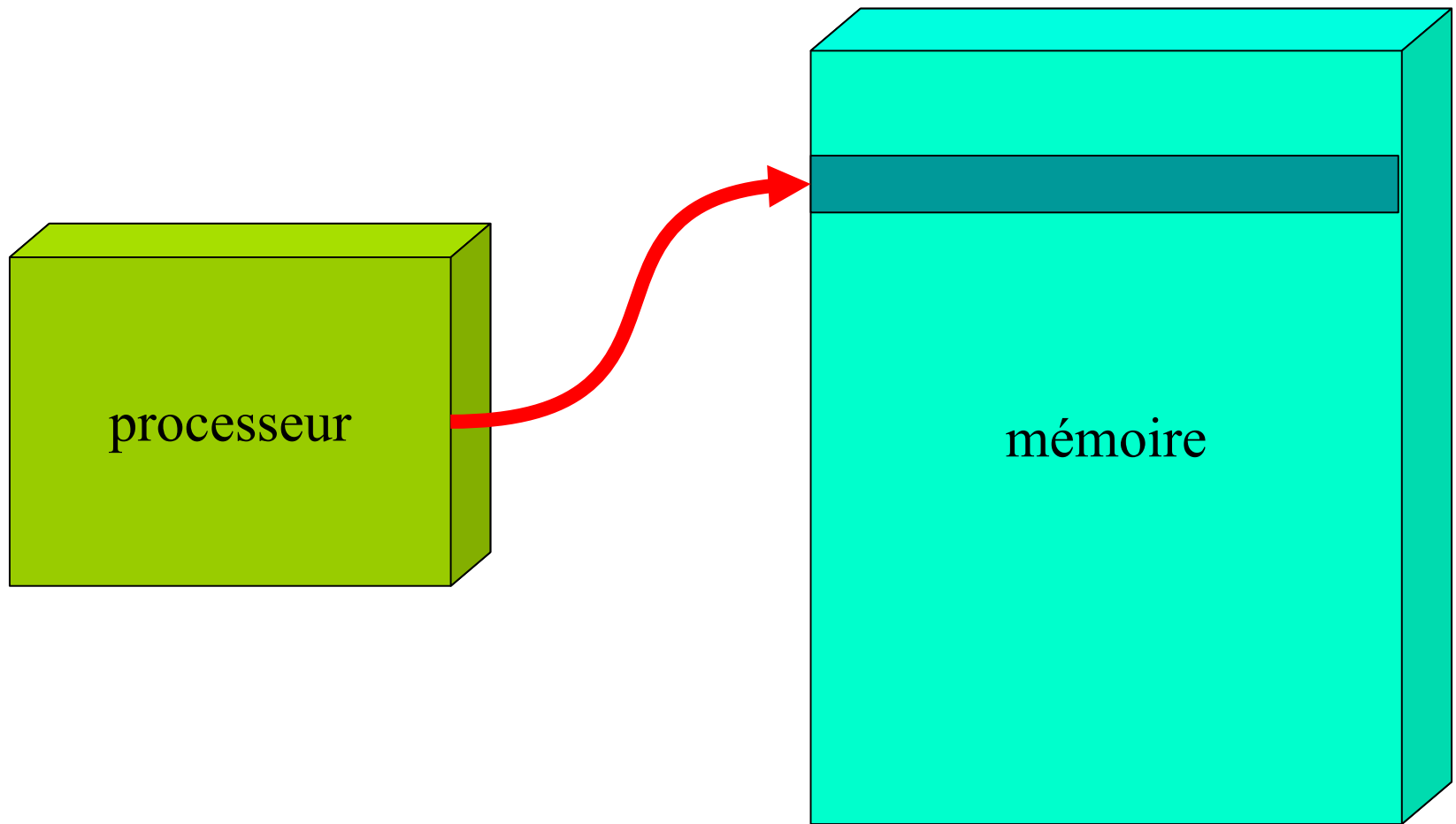
Temps de calcul

- Pour une entrée donnée :
 - Combien de temps prend chaque opération élémentaire ?
 - Combien de fois est exécutée chaque opération élémentaire ?
 - Profilation
- Pour ce faire, il faut disposer d'un modèle du calcul :
 - Modèle de la technologie utilisée pour réaliser l'algorithme
 - Décrit les ressources utilisées et leur coût
 - Doit être suffisamment réaliste
 - Doit faire abstraction des détails spécifiques de tel ou tel processeur

Modèle RAM

- RAM = Random Access Machine (c'est-à-dire, machine à mémoire adressable)
- Modèle théorique d'ordinateur, avec de nombreuses simplifications
- La machine RAM est néanmoins assez proche des ordinateurs actuels sur certains points :
 - Instructions exécutées en séquence
 - Chaque instruction prend un temps constant pour être exécutée
 - Le coût d'accès à n'importe quelle case de mémoire est constant

Mémoire adressable



Mesure des ressources

- Ressources utilisées pour le calcul d'une instance
- Temps :
 - nombre total des opérations élémentaires exécutées
 - mesure indépendante d'une machine particulière
 - simplification : temps égal pour chaque opération
- Espace :
 - quantité maximale d'informations à maintenir
 - inclut les données en entrée et les résultats
- Temps et espace sont corrélés
- $\text{espace} \leq k \cdot \text{temps}$

Analyse d'un algorithme

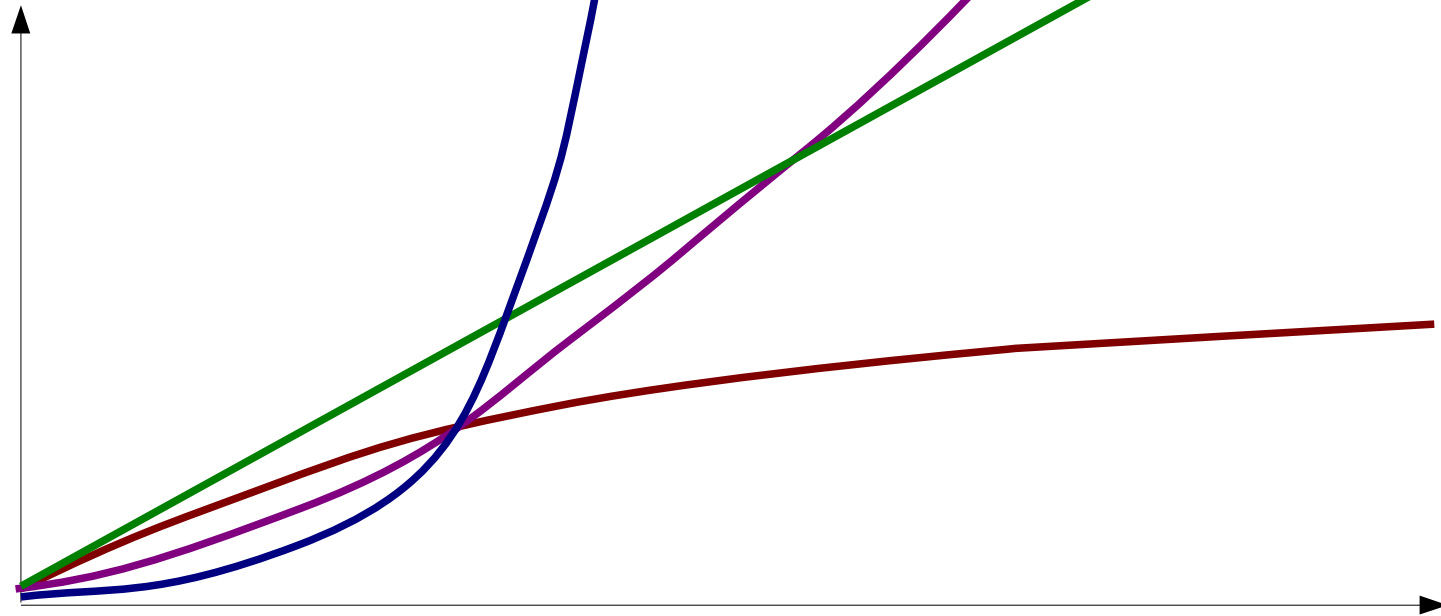
- Quantité de ressources est fonction des données en entrée
- Taille des données en entrée: n
- Meilleur cas: ex., le temps de calcul le plus court pour toutes les entrées de taille n .
- Cas moyen: ex., la moyenne des temps de calcul pour toutes les entrées de taille n .
- Pire cas: ex., le temps de calcul le plus long pour toutes les entrées de taille n .

Analyse pire cas

- Limite supérieure aux ressources que l'exécution de l'algorithme pourra jamais demander
- Le pire case s'avère assez souvent (ex. recherche d'une donnée inexistante)
- Dans la plupart des problèmes (et des algorithmes), le cas moyen est à peu près autant mauvais que le pire cas

Ordre

- Ce qui compte vraiment n'est pas la quantité précise des ressources demandées, mais comment celle-ci augmente en fonction de la taille des données en entrée.



Notations O , o , Θ et Ω

- Pour comparer des algorithmes, on fait une comparaison asymptotique de leurs fonctions de croissance du temps de calcul
- Notation “Grand O” introduite en 1894 par Paul Bachmann :
 $f(n) = O(g(n))$ signifie « il existe une constante C telle que, pour toute n , $|f(n)| \leq C|g(n)|$ »
- En tout, 4 notations asymptotiques sont souvent utilisées
 - $f(n) = O(g(n)) \equiv \exists C : \forall n, |f(n)| \leq C|g(n)|$
 - $f(n) = \Omega(g(n)) \equiv \exists C : \forall n, |f(n)| \geq C|g(n)|$
 - $f(n) = \Theta(g(n)) \equiv f(n) = O(g(n))$ et $f(n) = \Omega(g(n))$
 - $f(n) = o(g(n)) \equiv |f(n)|/|g(n)| \rightarrow 0$ pour $n \rightarrow +\infty$

Notations O , o , Θ et Ω

- Moins formellement (en termes d'ordres de croissance) :
 - $f(n) = O(g(n))$: l'ordre de f n'est pas plus que celui de g
 - $f(n) = \Omega(g(n))$: l'ordre de f est au moins celui de g
 - $f(n) = \Theta(g(n))$: f et g ont le même ordre de croissance
 - $f(n) = o(g(n))$: g approche l'infini plus vite que f
(f est négligeable devant g , g est prépondérante devant f)

Règles de manipulation de O

$$m \leq m' \rightarrow n^m = O(n^{m'}),$$

$$O(f(n)) + O(g(n)) = O(|f(n)| + |g(n)|),$$

$$f(n) = O(f(n)),$$

$$c \cdot O(f(n)) = O(f(n)),$$

$$O(O(f(n))) = O(f(n)),$$

$$O(f(n))O(g(n)) = O(f(n)g(n)),$$

$$O(f(n)g(n)) = f(n)O(g(n)).$$

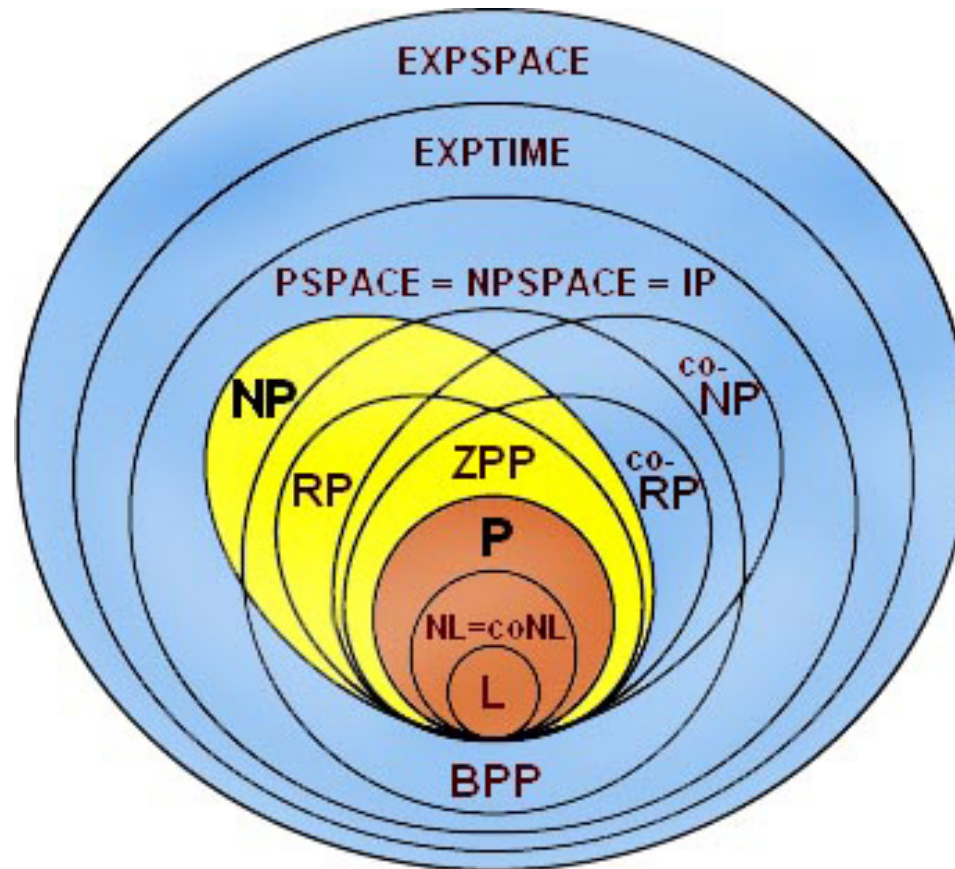
Classes de complexité

- $O(1)$ constante
- $O(\log n)$ logarithmique
- $O(\sqrt{n})$ racinaire
- $O(n)$ linéaire
- $O(n \log n)$ linéarithmique ou quasi-linéaire
- $O(n^2)$ quadratique
- $O(n^3)$ cubique
- $O(n^d)$ polynomiale de degré d
- $O(c^n)$ exponentielle
- $O(n!)$ factorielle
- $O(2^{2^n})$ doublement exponentielle

Classes de complexité

- En fait, il y a quatre familles de classes de complexité
 - **TIME(t(n))**, la classe des problèmes de décision qui peuvent être résolus en temps de l'ordre de grandeur de $t(n)$ sur une machine déterministe.
 - **NTIME(t(n))**, la classe des problèmes de décision qui peuvent être résolus en temps de l'ordre de grandeur de $t(n)$ sur une machine non déterministe.
 - **SPACE(s(n))**, la classe des problèmes de décision qui requièrent pour être résolus un espace de l'ordre de grandeur de $s(n)$ sur une machine déterministe.
 - **NSPACE(s(n))**, la classe des problèmes de décision qui requièrent pour être résolus un espace de l'ordre de grandeur de $s(n)$ sur une machine non déterministe.

Classes de complexité



Notion de pseudo-langage

- On a besoin d'un langage formel minimum pour décrire un algorithme
- Un langage de programmation (Python, Java, C, Pascal, etc.) est trop contraignant
- Dans la littérature, les algorithmes sont décrits dans un pseudo langage qui ressemble à un langage de programmation (le pseudo langage utilisé dépend donc de l'auteur et peut être spécifié par celui-ci en début d'ouvrage)

Pseudo-langage

- Tous les pseudo langages recouvrent les mêmes concepts
 - Variables, affectation
 - Structures de contrôle : séquence, conditionnelle, itération
 - Découpage de l'algorithme en sous-programmes (fonctions, procédures).
 - Structures de données simples ou élaborées (tableaux, listes, dictionnaires, etc.)

Pseudo-langage : variables, affectations

- Les variables sont indiquées avec leur type : booléen b, entier n, réel x, caractère c, chaîne s, etc.
- On est souple du moment qu'il n'y a pas d'ambiguïté
- Le signe de l'affectation n'est pas « = » comme en Python, ni « := » (comme en Pascal) mais « ← » qui illustre bien la réalité de l'affectation (« mettre dedans »)

Pseudo-langage : structures de données

- Les tableaux sont utilisés. Si A est un tableau, $A[i]$ est le i ème élément du tableau
- Les structures sont utilisées. Si P est une structure modélisant un point et x un champ de cette structure représentant l'abscisse du point, $P.x$ est l'abscisse de P

Remarque : une structure est une classe sans les méthodes

Pseudo-langage : *le séquençement des instructions*

- Les instructions simples sont séquencés par « ; » (si besoin)
- Les blocs d'instructions sont entourés par
 - { ... }, comme en C, C++ et Java
 - début ... fin, comme en Pascal
- Ou simplement distingués par la mise en forme (même niveau d'indentation), comme en Python

Pseudo-langage : la conditionnelle

La conditionnelle est exprimée par :

```
si condition
    instruction1
sinon
    instruction2
```

Pseudo-langage : les itérations

tant que condition
instruction

faire
instruction

tant que condition

répéter
Instruction

jusqu'à condition

pour i de min **à** max
instruction

Pseudo-langage : les fonctions

- `maFonction(↓ int i, ↑ int j, ⇕ int k);`
 - ↓ = en entrée : la fonction lit la valeur du paramètre, ici `i`. Les modifications qu'elle fera avec `i` ne seront pas transmises au programme appelant
 - ↑ = en sortie : la fonction ne lit pas la valeur du paramètre, ici `j`. Elle écrit dans `j` et le programme appelant récupère cette valeur, donc `j` peut être modifié par la fonction
 - ⇕ = en entrée/sortie : la fonction lit la valeur du paramètre `k`, et passe au programme appelant les modifications faites pour `k`
- Par défaut, on considérera que le paramètre est passé en entrée.
- Le passage en entrée/sortie est souvent appelé passage par référence ou par variable

Merci de votre attention

