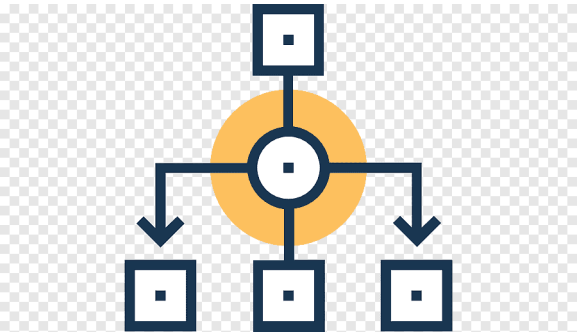
Algorithmique Avancée

1.0



Noureddine Amraoui
Maître assistant classe B
Université Mohamed Boudiaf de M'Sila

24 avril 2022

Table des matières

Objectifs

A l'issu de cet enseignement, l'apprenant sera capable de :

- En terme de Savoir :
  a. Maîtriser les bases de l'analyse algorithmique.
  b. Identifier les stratégies de résolution de problèmes.
  c. Connaître les classes de problèmes.
- En terme de Savoir-faire :
  a. Analyser et classer les problèmes de différents domaines.
  b. Construire la ou les solutions.
  c. Évaluer les différentes solutions en terme de calcul de complexité.
  d. Choisir la meilleure solution.
- En terme de Savoir-être :
  a. Vous sensibiliser à la résolution des problèmes.

I.Chapitre 1 : Complexité d'un algorithme, d'un problème

1. Pré-requis

L'étudiant doit connaître :

- Les bases de la programmation.
- Les base de l'analyse mathématique.

2. Pré-test

2.1. Exercice : Les bases de structure de données

[*Solution n°1 p 8*]

Lequel des éléments suivants est une structure de données non linéaire

○	Les piles.
○	Les listes.
○	Les chaîne de caractères.
○	Les arbres.

2.2. Exercice : Pouvez-vous définir un algorithme ?

[*Solution n°2 p 8*]

Qu'est-ce qu'un algorithme ?

○	Un organigramme.
○	Un organigramme ou un pseudo-code.
○	Instructions étape par étape utilisées pour résoudre un problème.
○	Une décision.

3. Algorithmes

Définition

On appelle **calcul** une séquence d'opérations élémentaires qui est :

1. finie.
2. déterministe, i.e. à chaque étape la nouvelle opération à effectuer est entièrement déterminée par les précédentes).
3. réalisable (chaque opération élémentaire peut être réalisée en un temps fini).
4. complète (les informations requises pour une étape de l'algorithme proviennent uniquement des étapes précédentes.

La définition ci-dessus n'est pas très formelle, mais elle est suffisante pour cette première approche. Pour une définition formelle d'un calcul, il faut se ramener à un modèle de calcul plus formel tel celui de la machine de Turing.

Lorsqu'un calcul s'arrête en un temps fini et que le résultat final fournit la réponse au problème on dit alors que ce calcul est un **algorithme**.

Lorsqu'il s'arrête en un temps fini mais que l'on n'a pas réussi à démontrer que le résultat final est le bon, on dit alors que ce calcul est une **heuristique**.

On cite souvent l'exemple d'une recette de cuisine comme exemple d'algorithme primitif. Au regard des définitions précédentes une recette de cuisine est certainement un calcul, mais la preuve formelle de la validité du résultat est difficile (surtout quand je suis au fourneau !), et donc il faut les ranger dans la catégorie des heuristiques.

En résumé avant d'analyser un algorithme il faut d'abord vérifier sa preuve, i.e. qu'il s'arrête en un temps fini et que son résultat est le bon.

4. Quelques paradigmes utiles pour construire un algorithme efficace

Méthode

Un paradigme est une méthode générique qui s'applique dans plusieurs situations algorithmiques. Voici une liste de quelques paradigmes algorithmiques :

- **L'induction :** Ce paradigme donne lieu a des processus itératifs.
- **Diviser pour régner :** Exemples : procédés dichotomiques, mais aussi d'une manière plus générale tous les algorithmes récursifs.
- **Trouver un ordre "optimal" sur les opérations à effectuer :** Exemples : les parcours en profondeur dans les graphes, les utilisations des ordres lexicographiques.
- **Utiliser une structure de données ad hoc :** Cette structure de données doit permettre de réaliser efficacement les opérations de base de l'algorithme. Exemples : arborescences binaires ordonnées de recherche, arborescences bicoloriés, B-arborescences, ?les de priorité.
- **Stocker des résultats des calculs intermédiaires :** Ici intervient le compromis calcul/mémoire. Par exemple pour calculer un sinus on peut utiliser un calcul (développement en série) ou aller chercher une valeur dans une table (si le calcul a déjà été fait).
- **Utiliser un procédé stochastique (à base de tirages aléatoires) :** Les algorithmes randomisés ou probabilistes sont souvent très simples à mettre en œuvre et l'on peut montrer qu'ils donnent le bon résultat avec une forte probabilité[1].

5. Complexité d'un algorithme

Fondamental

L'analyse d'algorithmes se veut un outil d'aide à la comparaison des performances des algorithmes. Nous utiliserons ci-après le mot complexité à son sens premier (du latin complexus enlacé, embrassé signifiant un objet constitué de plusieurs éléments ou plusieurs parties).

Les outils et méthodes que nous allons développer dans ce cours devrait vous convaincre que la complexité des algorithmes n'est pas compliquée (qui est hélas le sens commun actuel du mot complexité) !

Dans le cadre de l'analyse d'algorithmes, il s'agit de mesurer les ressources critiques (coûteuses) utilisées par les algorithmes. Parmi les ressources fréquemment étudiées on trouve : le temps d'exécution ou de programmation, la mémoire, les processeurs, les messages, mais on pourrait tout aussi bien étudier sur les implémentations matérielles d'un algorithme la surface du circuit ou le nombre de portes logiques, l'énergie consommée, les radiations émises . . . ou encore le nombre de prions libérés (informatique biologique).

D'ailleurs lors de la conception de certains ordinateurs mobiles le problème crucial est celui de l'énergie (i.e. la consommation électrique) induite

par l'émission réception. Le calcul proprement dit engendrant une consommation négligeable. Il faut donc trouver des protocoles où l'on émet-écoute

le moins longtemps possible (à coup sûr).

Il s'agit d'évaluer les performances de l'algorithme en fonction de cette ressource, i.e. la taille de cette ressource utilisée par l'algorithme.

Définition

On appelle complexité de l'algorithme A pour la ressource R la fonction :

T(A, R, n) = max{ ressource R utilisée par A sur l'ensemble des données de taille n}

Définition

La taille d'une donnée est simplement la taille d'un bon codage en mémoire de cette donnée exprimée en nombres de bits.

Ainsi la taille d'un codage d'un entier n sera dlog2(n + 1)e, et non pas n. On préfère les codages binaires aux codages unaires des entiers.

Lorsqu'il n'y a pas d'ambiguïté, on notera T(A, R, n) simplement par T(n). Cette mesure est appelée la mesure du plus mauvais cas, car elle garantit que tout comportement de l'algorithme A utilisera au plus T(A, R, n) éléments de la ressource R

6. Exercices

6.1. Exercice : Avez-vous maîtriser la complexité d'un algorithme ?

[*Solution n°3 p 8*]

La complexité d'un problème est celle du :

○	Meilleur algorithme qui le résout.
○	Pire algorithme qui le résout.
○	Moyen algorithme qui le résout.

6.2. Exercice : Pouvez-vous identifier un algorithme efficace ?

[Solution n°4 p 8]

Un algorithme est dit efficace si sa complexité est :

○	Linéaire.
○	Polynomiale.
○	Quasi-linéaire.
○	Quadratique.

Solution des exercices

> Solution n°1 *(exercice p. 4)*

Lequel des éléments suivants est une structure de données non linéaire

○	Les piles.
○	Les listes.
○	Les chaîne de caractères.
◉	Les arbres.

> Solution n°2 *(exercice p. 4)*

Qu'est-ce qu'un algorithme ?

○	Un organigramme.
○	Un organigramme ou un pseudo-code.
◉	Instructions étape par étape utilisées pour résoudre un problème.
○	Une décision.

> Solution n°3 *(exercice p. 6)*

La complexité d'un problème est celle du :

◉	Meilleur algorithme qui le résout.
○	Pire algorithme qui le résout.
○	Moyen algorithme qui le résout.

> Solution n°4 *(exercice p. 7)*

Un algorithme est dit efficace si sa complexité est :

○	Linéaire.
◉	Polynomiale.
○	Quasi-linéaire.
○	Quadratique.

Références

[1] R. Motvani and P. Raghavan. Randomized Algorithms. Cambridge University Press, 1995.