Complexité

Nicolas Bourras

September 14, 2020

Contents

0.1 Premier cours

10 Septembre 2020, 8h

0.1.1 Informations utiles

Professeur principal :
Philippe Jégou / `philippe.jegou@univ-amu.fr`
Volume horaire :

- 10 séances de 2h de cours; premiers 5 avec jégou, dernier 5 avec Porreca
- 10 séances de 2H de TD;
- 7 séances de 2h;

Le prof va beaucoup utiliser ametice. Pour envoyer un message, commencer l'objet par "[M1-Info-Complexite]"
Evaluation :

- Session 1 : $0.75 * max((CC + (2 * ET))/3, ET) + 0.25 * TP$ avec CC: contrôle continu a priori avec un partiel à la "mi-temps" avec TP: petits projets réalisés en groupes pendant les TP
- Session 2 : $max(ET, (0.75 * ET) + 0.25 * TP)$

0.1.2 Connaissances requises

- Algorithmique
  - conception d'algorithmes
  - analyse de la complexité (srtout pire des cas et "grand O")
  - algorithmique des graphes
- Théorie de Langages Formels :
  - Notions de base : alphabet, mots, langages, etc..
  - automates finis
  - machines de turing
- Logique des propositions
  - notions de base: variables, clauses, interprétation, CNF
  - traitement de formules: quine, dpll, résolution

0.1.3 Références

-> Computers and Interactability - A guide to the Theory of NP-Completeness. Freeman And Co., 1979

0.1.4 Plan

De quoi on va parler?

-> estimer le temps d'exec d'un algo -> déterminer la difficulté d'un problème (sa complexité) -> $P! = NP$?

Objectifs

-> identifier les problèmes polynomiaux -> identifier les problèmes non-polynomiaux, et évaluer leur difficulté

plan

- complexité, présentation et introduction
- rappels d'algorithmique et de complexité
- problèmes et complexité des problèmes
- cadre formel
- Classe P
- classe NP (changement de prof)
- ...

0.1.5 Rappels, algorithmique et complexité

-> analyse de complexité -> analyse des graphes

Complexité

objectif : estimer les temps de calcul, ne pas perde de temps à implémenter un algo qui ne sera pas efficace, concevoir les algos les plus rapides avant de les implémenter, ne pas être limité par une machine
on va introduire un modèle pour estimer le temps d'execution d'un programme

modèle principe 1 : actions élémentaires exécutables en temps constant
on identifie: - les affectations - les opérations arithmétiques - les tests - les branchements
Pour simplifier, on suppose que chacune de ses instuctions prennent le même temps. De plus, on ignore le cout des branchements.
principe 2 : l'analyse s'opère en fonction de la taille de la donnée en entrée
La taille est par exemple la taille d'un tableau, ou la taille du codage d'un entier.
principe 3 : l'analyse prend on compte le comportement asymptotique du temps
-> comment évolue le temps de calcul quand la taille de la donnée s'acroit
-> O(n) -> theta(n)
principe 4 : pire des cas

- l'analyse de la complexité en moyenne n'est pas facile (comment créer un jeu de donnée?); de plus elle n'est pas pertinente pour l'UE

- analyse lisse d'algorithme permet d'éviter les pb de l'analyse de la complexité en moyenne, mais elle n'est pas pertinente non plue
- l'analyse de la complexité dans le meilleur des cas ne l'est pas non plus

Récap Rappel : O est une majoration -> insérer formule O O(f(n)) = {g: N -> R* : } ...
et θ est exact -> insérer formule theta(f(n)) = {g: N -> R* : } ...
et grand ω est la minoration (pas à savoir)

Rappels sur le codage Codage unaire: n bits Codage binaire: log2(n) + 1
Certains algos, comme celui (naif) pour vérifier un nombre premier, est exponentiel à cause du codage.

Validité Le modèle n'est pas forcément valide dans tous les cas, il n'est pas forcément assez précis.
Dans le chapitre 4 (cadre formel), on introduira un modèle plus précis (qui se base sur une machine de turing).