Chapitre 08 : Complexité

07 novembre

Après nous être intéressé aux problèmes de décision que l'on peut ou non résoudre à l'aide d'un programme informatique, nous allons ici nous concentrer sur les ressources nécessaires à cette résolution. Ce chapitre ne portera que sur la complexité temporelle c'est-à-dire sur les ressources de temps que le programme devra nécessairement utiliser afin de résoudre le problème. Une autre ressource fondamentale est l'espace mémoire nécessaire mais ce sujet n'est pas abordé par le programme de MPI.

La notion de complexité est ici abordée dans une perspective différente de celle régulièrement utilisée jusqu'ici. En effet, lorsque nous parlons de complexité on parle en général de la complexité d'un algorithme ou d'un programme c'est-à-dire que l'on analyse a postériori les ressources utilisées par notre programme. Ici, on étudie la complexité d'un problème c'est-à-dire que l'on met en évidence la quantité de ressource nécessaire à la résolution du problème indépendamment d'un programme ou d'un algorithme concret le résolvant. On cherche donc à classifier les problèmes par degré de difficulté intrinsèque.

1 Problème de décision-Complexité

On rappelle que la donnée d'un **problème de décision** correspond à la donnée d'une fonction booléenne totale définie sur un ensemble D dénombrable.

Dans un programme, on appelle **opération élémentaire** une opération dont l'exécution est en temps constant. La lecture et l'écriture en mémoire sont des opérations élémentaires tout comme les opérations arithmétiques sur des entiers représentés en machine ou les comparaisons de leurs valeurs. On remarque que la manipulation arithmétique d'entiers dont la valeur dépasse la capacité mémoire n'est pas considérée comme une opération élémentaire.

La **taille de l'entrée** est l'espace mémoire nécessaire à son stockage et dépend donc de son mode de représentation. Nous travaillons cependant avec des ordres de grandeur et non pas avec le nombre exact de bits utilisés donc une connaissance précise de la représentation n'est pas fondamentale.

Par exemple, pour un graphe $G = (S, A)$, la taille de l'instance sera $O(|S|^2)$ si on utilise une matrice d'adjacence, $(|S| + |A|)$ avec des listes d'adjacence et $O(|A|)$ avec une liste des arêtes. Cependant on peut toujours dire que la taille d'un graphe est $O(|S|^2)$. Pour un entier représenté en machine, la taille est une constante (32 ou 64 bits en général) suivant le type et l'architecture mais pour un entier n arbitrairement grand sa taille sera $O(\log(n))$ qui correspond à la taille mémoire nécessaire au stockage de sa représentation binaire. Pour une liste ou un tableau d'entiers "machine", la taille sera sa longueur.

La **complexité d'un algorithme** est le nombre d'opérations élémentaires qu'il effectue dans le pire cas. C'est donc le maximum sur toutes les entrées possibles du nombre d'opérations qu'il effectue sur cette entrée donnée. On exprime cette complexité comme un ordre de grandeur de la taille n de son entrée.

Soit $t : \mathbb{N} \to \mathbb{N}$. On dit qu'un problème de décision est dans la **classe de complexité** $DTIME(t(n))$ s'il admet un algorithme, dont la complexité est en $O(t(n))$, qui le résout.

2 La classe P

On dit qu'un problème de décision appartient à la classe **P** s'il existe une fonction polynomiale t de la taille de l'entrée et un algorithme A dont la complexité est $O(t(n))$ sur une entrée de taille n.

Voici quelques exemples de problèmes qui sont dans **P** :

— Déterminer si un tableau admet ou non des doublons.
— Déterminer si un graphe est connexe.
— Déterminer si le langage reconnu par un automate est vide.
— Déterminer si deux entiers sont premiers entre eux.
— ...

Remarques importantes : on notera que ce sont bien des problèmes et non pas des algorithmes qui sont dans la classe **P**. De plus, cette notion ne porte que sur les problèmes de décision et n'est pas vouée à être étendue à d'autres types de problèmes algorithmiques.

Le fait que le degré des fonctions polynomiales mises en évidence est quelconque permet de faire des estimations assez larges de complexité pour garantir qu'un problème est dans la classe **P**.

La classe **P** est considérée comme une classe de problèmes dont la résolution algorithmique est "raisonnable" en temps mais cette affirmation n'est que théorique car, si le degré est trop grand, un problème de la classe P peut parfaitement amener à des algorithmes dont le temps d'exécution ne permet pas de l'utiliser en pratique.

3 La classe NP

Définition 1. *Soit $\mathcal{P}$, un problème de décision associé à : $f : D \to \mathbb{B}$. On dit que le problème $\mathcal{P}$ est vérifiable si on dispose d'une fonction $v : D \times \{0,1\}^* \to \mathbb{B}$ de vérification tels que $\forall x \in D, f(x) = Vrai$ ssi $\exists c \in \{0,1\}^*, v(x,c) = Vrai$.*

Définition 2. *Un problème de décision $\mathcal{P}$ est dans la classe **NP** ssi*

1. *Il est vérifiable avec un certificat dont la taille est polynomiale en la taille de l'entrée associée.*
2. *la fonction de vérification correspond à un problème de décision de la classe **P**.*

Exemple 1. *Les exemples de problèmes suivants sont dans la classe **NP** :*

1. *Savoir si une formule de la logique propositionnelle est satisfiable.*
2. *Savoir si un graphe est 3-coloriable.*
3. *Savoir si une grille de Sudoku admet une solution.*
4. *Savoir si un entier n'est pas premier.*
5. *Savoir si un graphe admet un chemin hamiltonien.*
6. *Savoir si graphe admet une clique de taille au moins k, avec k et G comme entrées.*
7. *...*

Théorème 1. ***P*** $\subset$ ***NP***.

4 Réduction polynomiale

Définition 3. *Soient deux problèmes de décision associés aux fonctions $f_1 : D_1 \to \mathbb{B}$ et $f_2 : D_2 \to \mathbb{B}$. On dit que f_1 se réduit polynomialement à f_2, et on note $f_1 \leq_P f_2$ s'il existe une fonction $g : D_1 \to D_2$ calculable en temps polynomial (la complexité d'un algorithme la calculant peut être en temps polynomial) telle que $\forall x \in D, f_1(x) = f_2(g(x))$.*

Théorème 2. *Soient deux problèmes de décision associés aux fonctions $f_1 : D_1 \to \mathbb{B}$ et $f_2 : D_2 \to \mathbb{B}$. On suppose que $f_1 \leq_P f_2$ et que $f_2 \in$ **P** alors $f_1 \in$ **P**.*

Exemple 2. *Le problème de tester si une liste d'entiers positifs est triée par ordre croissant se réduit polynomialement au problème de tester si une liste d'entiers négatifs est triée par ordre décroissant.*

*On définit le problème de décision **Clique** suivant : étant donnés un graphe G et un entier k, existe-t-il une clique de taille au moins k dans G ?*

*On va montrer que $3-SAT$ se réduit à **Clique**.*

5 Le théorème de Cook-Levin

Définition 4. *Un problème de décision est dit NP-complet s'il vérifie les deux propriétés suivantes :*

1. *il appartient à la classe **NP**.*
2. *tout problème de décision de la classe **NP** se réduit polynomialement à lui.*

Théorème 3. *Si un problème est NP-complet et est dans **P** alors **NP** = **P**.*

Théorème 4. *Si un problème A est NP-complet et qu'il se réduit polynomialement à un problème $B \in$ **NP** alors B est NP-complet.*

Théorème 5. *CNF-SAT est NP-complet. (théorème de Cook-Levin).*

admis

Théorème 6. *3-SAT est NP-complet.*

Théorème 7. *Clique est NP-complet.*

Théorème 8. *Independant Set est NP-complet.*

Théorème 9. *Vertex Cover est NP-complet.*

6 Problème d'optimisation-problème de recherche

Dans les deux derniers chapitres, on s'est restreint aux problèmes de décision. Cependant il existe deux autres grandes familles de problèmes informatiques : les problèmes de recherche et les problèmes d'optimisation. Chacune des ces familles de problèmes pourra se ramener à des problèmes de décision d'où l'importance de ces derniers.

Définition 5. *Un problème de recherche sur un domaine d'entrées E et un domaine de solutions S est défini par une relation $R \subset E \times S$. On dit qu'un algorithme résoud ce problème lorsque pour toute entrée $x \in E$, l'algorithme renvoie $y \in S$ tel que $R(x, y)$.*

Définition 6. *Soit un problème de recherche associé à un relation $R \subset E \times S$. On peut lui associer deux types de problèmes de décision :*

1. *Le problème d'existence $f_\exists : E \to \mathbb{B}$ telle que $f_\exists(x) = Vrai$ ssi il existe $s \in S$ tel que $R(x, s)$.*
2. *Le problème de vérification $v : E \times S \to \mathbb{B}$ telle que $v(x, e) = Vrai$ ssi $R(x, s)$.*

Définition 7. *Un problème d'optimisation sur un domaine d'entrées E et un domaine de solutions S est défini par une relation $R \subset E \times S$ et une fonction de coût $c : S \to \mathbb{R}^+$. L'objectif d'un tel problème est ,étant donné un $x \in E$, de trouver $s \in S$ tel que $R(x, s)$ de coût $c(s)$ minimal (ou maximal).*

Un algorithme résoudra ce problème si sur l'entrée x il renverra un tel s.

Définition 8. *Soit un problème d'optimisation fixé défini par $R \subset E \times S$ et $c : S \to \mathbb{R}^+$.*

Si on fixe un seuil $c_0 \in \mathbb{R}^+$ alors on peut définir un problème de décision $f_{c_0} : E \to \mathbb{B}$ par $f_{c_0}(x) = Vrai$ ssi $\exists s \in S,$ t.q. $c(s) \leq c_0$ et $R(x, s)$.

Exemple 3. *Donner un problème de décision associé au problème d'optimisation de la recherche d'un plus court chemin entre deux sommets dans un graphe.*

9 Décidabilité et classes de complexité S3-4

On s'intéresse à la question de savoir ce qu'un algorithme peut ou ne peut pas faire, inconditionnellement ou sous condition de ressources en temps. Cette partie permet de justifier la construction, plus haut, d'algorithmes exhaustifs, approchés, probabilistes, etc. On s'appuie sur une compréhension pratique de ce qu'est un algorithme.

Notions	**Commentaires**
Problème de décision. Taille d'une instance. Complexité en ordre de grandeur en fonction de la taille d'une instance. Opération élémentaire. Complexité en temps d'un algorithme. Classe **P**.	Les opérations élémentaires sont les lectures et écritures en mémoire, les opérations arithmétiques, etc. La notion de machine de Turing est hors programme. On s'en tient à une présentation intuitive du modèle de calcul (code exécuté avec une machine à mémoire infinie). On insiste sur le fait que la classe **P** concerne des problèmes de décision.
Réduction polynomiale d'un problème de décision à un autre problème de décision.	On se limite à quelques exemples élémentaires.
Certificat. Classe **NP** comme la classe des problèmes que l'on peut vérifier en temps polynomial. Inclusion $\mathbf{P} \subseteq \mathbf{NP}$.	Les modèles de calcul non-déterministes sont hors programme.
NP-complétude. Théorème de Cook-Levin (admis) : SAT est NP-complet.	On présente des exemples de réduction de problèmes NP-complets à partir de SAT. La connaissance d'un catalogue de problèmes NP-complets n'est pas un objectif du programme.
Transformation d'un problème d'optimisation en un problème de décision à l'aide d'un seuil.	
Notion de machine universelle. Problème de l'arrêt.	

Mise en œuvre
On prend soin de distinguer la notion de complexité d'un algorithme de la notion de classe de complexité d'un problème. Le modèle de calcul est une machine à mémoire infinie qui exécute un programme rédigé en OCaml ou en C. La maîtrise ou la technicité dans des formalismes avancés n'est pas un objectif du programme.