Chapitre III

Complexité des problèmes d'ordonnancement

INTRODUCTION

Les premières applications scientifiques des ordinateurs portaient sur des données de taille réduite ; il n'était donc pas indispensable de se préoccuper du nombre d'opérations des algorithmes utilisés. Depuis, le champ des applications s'est considérablement élargi : on cherche à résoudre soit des problèmes hautement combinatoires en recherche opérationnelle (emploi du temps, tournées, ordonnancements ...), soit des problèmes de taille de plus en plus grande en analyse numérique (résolution d'équations aux dérivées partielles ...). On s'est vite aperçu que, malgré la rapidité des ordinateurs, il y avait des limites qu'il fallait cerner.

Une première étape a été d'évaluer en fonction de la taille des données le nombre d'opérations des algorithmes. Nous expliquons comment évaluer cette complexité et montrons l'importance fondamentale des algorithmes polynômiaux. Malheureusement, pour un grand nombre de problèmes, on n'a pas réussi à construire un algorithme polynômial, ni à prouver qu'il n'en existait pas.

Une deuxième étape a été de classer les problèmes combinatoires en deux sous-ensembles : les problèmes résolus par un algorithme polynômial et les problèmes NP-difficiles. Ces derniers sont liés par la propriété suivante : si un seul problème NP-difficile est polynômial, alors la plupart des problèmes réputés difficiles sont également polynômiaux. Aussi les chercheurs pensent-ils que si un problème est NP-difficile il n'existe pas d'algorithme polynômial le résolvant ; c'est la conjecture fondamentale de la théorie de la complexité. Nous rappelons cette théorie.

Ceci explique pourquoi, lors de l'étude d'un problème d'ordonnancement, on commencera par chercher à classer le problème. Si on parvient à montrer qu'il est polynômial, le problème sera résolu ; la deuxième partie de ce livre est consacrée à la présentation d'algorithmes polynômiaux. Si on parvient à montrer qu'il est NP- difficile, il restera à mettre au point des méthodes de traitement adaptées ; la troisième partie du livre est consacrée à la présentation de ces méthodes.

I. COMPLEXITE DES ALGORITHMES

Introduction

On pourrait penser que la complexité d'un algorithme ne peut être évaluée indépendamment de l'ordinateur et du langage informatique choisi. Il n'en est rien, du moins pour les machines monoprocesseurs (1). En effet, le modèle mathématique des machines de Turing représente, à des coefficients de rapidité près, le comportement des ordinateurs. Dans ce paragraphe, nous introduisons les machines de Turing ; nous montrons leur fonctionnement sur un exemple ; nous expliquons pourquoi elles permettent de définir la complexité des algorithmes et nous montrons l'importance des algorithmes polynômiaux.

I.1. Les machines de Turing

Définition

Une machine de Turing est constituée d'une unité centrale (UC), d'une tête de lecture et d'écriture et d'une bande infinie dont les cases sont indicées par les éléments de $\mathbb{N}$.

Un programme M est constitué d'un alphabet Γ contenant le symbole blanc ($\Gamma = X_U \{b\}$), d'un ensemble fini Q contenant un état q_0 dit initial et un état q_y dit final, et d'une fonction δ de $(Q-\{q_y\}) \times \Gamma$ dans $Q \times \Gamma \times \{-1, +1\}$ dite de transition.

(1) Pour les machines multiprocesseurs (SIMD, MIMD et systèmes distribués), il n'y a pas de modèle mathématique satisfaisant.

A l'instant 0, un mot $x=x_1,x_2,\ldots,x_n$ de X^* est placé sur la ande infinie dans les cases 1,2,...,n. Les autres cases sont emplies par des blancs. La tête de lecture-écriture est placée en ace de la case 1. L'état de la machine est q_o (figure 1).

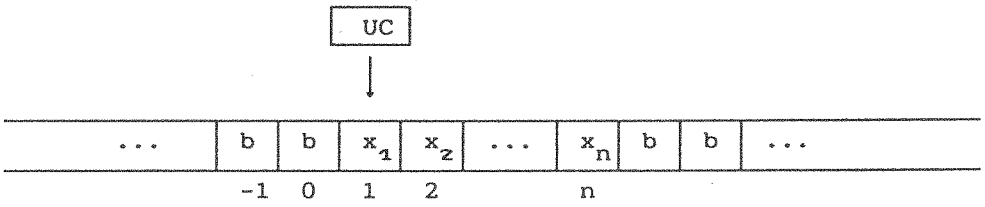

figure (1)
Initialisation de la machine de Turing.

arche du programme

Le programme marche à l'aide de la fonction de transition δ. l'instant t, la machine est en un certain état q, la tête de ecture-écriture est en face d'une case d'adresse a qui contient e symbole γ. Si $q=q_y$, la machine est définitivement arrêtée; inon, $\delta(q,\gamma) = (q',\gamma',\varepsilon)$ définit le changement d'état : à 'instant t+1, elle est dans l'état q' ; la tête de lecture-criture est en face de la case a+ε ; le contenu de la case 'adresse a est γ'.

angage reconnu

Le langage $L\subseteq X^*$ reconnu par le programme M est l'ensemble es mots pour lesquels la machine de Turing s'arrête en un temps ini.

lgorithme

Un algorithme est un programme pour lequel : $L=X^*$.

urée du programme pour une donnée x

Si la machine de Turing atteint l'état q_y en un temps fini

Exemple

—
—
—
—
—

q_o

q_o

q_o

q_o

q_o

q_o

q_1

Exemple

$\Gamma = \{0,1,b\}$ $Q = \{q_o, q_1, q_2, q_3, q_y\}$

δ	0	1	b
q_o	$(q_o, 0, 1)$	$(q_o, 1, 1)$	$(q_1, b, -1)$
q_1	$(q_2, 0, -1)$	$(q_3, 1, -1)$	$(q_3, b, -1)$
q_2	$(q_y, 0, 0)$	$(q_3, 1, -1)$	$(q_y, b, -1)$
q_3	$(q_3, 0, -1)$	$(q_3, 1, -1)$	$(q_3, b, -1)$

figure (2)

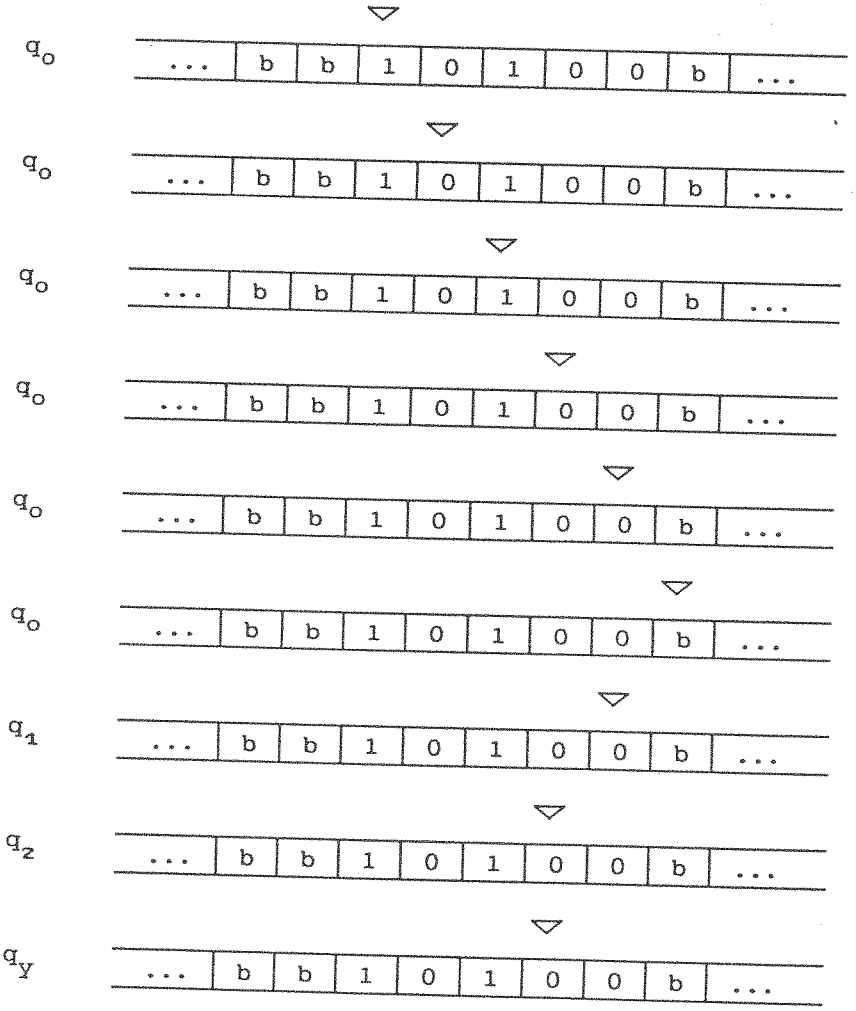

figure (3)
Configurations successives de la machine de Turing.

La figure 2 rapporte un programme reconnaissant le langage des nombres multiples de 4 écrits en base 2. La figure 3 décrit les configurations successives de la machine de Turing pour la donnée x = 10100. Le programme fonctionne de la façon suivante : la tête de lecture écriture "avance" tant qu'on n'a pas rencontré de blanc, et la machine de Turing reste en l'état q_0. Quand on rencontre un blanc la machine passe en l'état q_1 et la tête de lecture-écriture va vers l'arrière. Si le dernier caractère n'est pas un zéro, la machine passe en l'état q_3 : le chiffre n'est pas divisible par 4; sinon, elle passe en l'état q_2. Si l'avant dernier caractère est un 1, elle passe en l'état q_3 ; sinon elle passe en l'état q_y. Pour cet exemple, le nombre d'itérations du programme pour un mot x du langage est n+3.

Taille d'une donnée, complexité du programme

On appelle taille d'une donnée x la longueur n=|x|. On dit qu'un programme est de complexité O(q(n)), si il existe k tel que pour toute donnée x∈L de taille n≥1 ; on ait : T(x)≤k q(n).

On dira qu'un algorithme est polynômial d'ordre p si $q(n)=n^p$.

Exemple

Le programme précédent est de complexité O(n).

Machine de Turing et ordinateur

On montre que toute instruction du langage machine d'un ordinateur monoprocesseur peut être simulée sur une machine de Turing par un programme de complexité O(1). Il en est de même pour toute instruction élémentaire d'un langage évolué (addition, multiplication, test ...) [AHO 74].

Nous appellerons opération élémentaire toute instruction d'un langage informatique. Une opération élémentaire pourra donc être simulée sur une machine de Turing avec une complexité O(1) et s'exécutera sur un ordinateur en une durée majorée par un nombre fonction de la rapidité de cet ordinateur. On pourra donc évaluer un algorithme en comptant le nombre d'opérations élémentaires.

I.2. Evaluation des algorithmes

Introduction

Dans un algorithme, il y a en général une phase de lecture des données et le nombre d'opérations effectuées par cet algorithme est en relation avec la taille de ces données. Dans ce paragraphe, nous montrons comment calculer numériquement cette taille et comment évaluer en fonction de celle-ci la complexité d'un algorithme.

Choix d'un codage

Nous utiliserons les codages les plus compacts possibles. C'est pourquoi, un nombre a sera codé par son expression binaire et occupera une place $[\log_2 a](1)$. Cette remarque préliminaire est importante car si un nombre était codé par son expression unaire, il occuperait une place a. Or ceci pourrait cacher une exponentielle; par exemple : $a = 2^{32}$!

Taille d'un énoncé

Une fois choisi un codage, on peut associer à un énoncé un mot d'un langage. On appelle taille de l'énoncé, la longueur de ce mot. Par exemple, si on recherche le minimum de n nombres $a_1, a_2, \ldots, a_n$, on peut associer à ces n nombres le mot $x = n;a_1;a_2;\ldots;a_n$ défini sur l'alphabet $X = \{0,1,;\}$ où les n+1 nombres sont codés en binaire. La longueur du mot x est $(n+[\log_2 a_1] + \ldots + [\log_2 a_n] + [\log_2 n])$; c'est la taille de l'énoncé. Si nous supposons en outre que les n nombres $a_1, a_2, \ldots, a_n$ sont majorés par un nombre K, la taille de l'énoncé est en O(n). Cette hypothèse est souvent raisonnable car un mot mémoire a une longueur fixe.

Evaluation d'un algorithme

Pour déterminer l'efficacité d'un algorithme, nous évaluerons, en fonction de la taille d'un énoncé, le nombre d'opérations élémentaires nécessitées par cet algorithme dans le plus mauvais cas. Il est également parfois possible d'évaluer ce nombre en moyenne. Nous expliquons ci-dessous sur un exemple comment procéder.

(1) $[\log_2 a]$ est le plus petit entier supérieur à $\log_2 a$.