[Info 528]Mathématiques pour l'informatique

Paul CLAVIER

12 novembre 2013

Table des matières

1 Complexité

1.1 Pourquoi, Comment ?

But : prévoir le temps d'exécution d'un programme en fonction des paramètres.
On utilise des approximations pour :
- ne pas prendre en compte la vitesse du processeur.
- on ne sait pas calculer exactement le nombre d'opérations.

Les approximations ont du style : "La complexité est proportionnelle à ..."

1.2 Notation "O"

Définition : on dit que "f est un grand O de g"

Notation : $f = O(g) \Leftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = C$

Propriétés :
- si $f = O(g)$ et $f' = O(g)$ alors $f + f' = O(g)$
- si $f = O(g)$ et $\alpha \in \mathbb{R}^+$ alors $\alpha \cdot f = O(g)$
- si $f = O(g)$ et $g = O(h)$ alors $f = O(h)$
- si $\alpha \leq \beta$ alors $x^\alpha = O(x^\beta)$

1.3 Diviser pour régner

L'idée : pour résoudre un problème "gros" :
1. on découpe le problème en sous problèmes
2. on résout chaque sous problème (de la même manière)
3. on récolte les solutions

Complexité : la complexité est donnée par une formule de récurrence Si on divise un problème de taille n en sous-problèmes de taille $\frac{n}{b}$, $C(n) = a \times C\left(\frac{n}{b}\right) + f(n)$ où $f(n)$ est la complexité pour recoller les morceaux.

1.4 Résolutions des récurrences

1.4.1 Première méthode

- On commence par deviner le résultat
- On prouve le résultat par récurrence

1.4.2 Théorème fondamental

Si on a une complexité

$$C(1) = k$$
$$C(n) = a \times C\left(\frac{n}{b}\right) + f(n), a, b \in \mathbb{N}$$

La complexité est
- si $f(n) = O\left(n^{\log_b(a)-\epsilon}\right)$, alors

$$C(n) = \Theta\left(n^{log_b(a)}\right)$$

- si $f(n) = \Omega\left(n^{\log_b(a)+\epsilon}\right)$, alors
$$C(n) = \Theta(f(n))$$
- si $f(n) = \Theta\left(n^{\log_b(a)}\right)$, alors
$$C(n) = \left(\log(n) \times n^{\log_b(a)}\right)$$

2 Codes correcteurs d'erreurs, application aux QR codes

2.1 Introduction

Situation générale : on transmet un signal (suite de bits) sur un canal de communication non parfait. Le signal reçu est (légèrement) différent du signal envoyé.

But : ajouter de la redondance d'information pour repérer et corriger ces erreurs. On veut faire ceci de manière contrôlée.

Objectif : rajouter le moins de redondance possible en détectant le plus d'erreurs possible.

2.2 Définition

Un code de taille n est un ensemble de mots de n bits. La distance d'un code est le nombre de bits minimal qui permet de passer d'un mot du code à un autre mot du code.

2.3 Les codes linéaires

Un code est linéaire si le xor de mots corrects est correct.

3 Code de Hamming

Les codes linéaires sont générés par des matrices.

$$\text{Code de Hamming :} M = \begin{bmatrix} 1 & 0 & 0 & 0 & \vdots & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & \vdots & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & \vdots & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & \vdots & 1 & 1 & 1 \end{bmatrix}$$

Les mots du code s'obtiennent en additionnant les lignes de M (matrice génératrice). Les matricesgénératrices permettent de coder des mots arbitraires.
Pour vérifier si un mot et correct, on utilise une matrice de parité

$$\text{parité de Hamming :} H = \begin{bmatrix} 0 & 1 & 1 & 1 & \vdots & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & \vdots & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & \vdots & 0 & 0 & 1 \end{bmatrix}$$

Théorème : Le résultat de $H \times \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$ est $\begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$ si le mot est correct.

Pour décoder : Si le mot est correct, le mot initial est la concaténation des 4 premiers bits. sinon, il existe toujours une matrice de décodage qui permet de décoder