Option Informatique en Spé MP et MP*

Devoir surveillé du 3 décembre 2002

***

Résumé

Ce texte est conçu comme les sujets des concours communs. Il est organisé en trois exercices. Les deux premiers vous permettront de réviser vos connaissances sur la logique et les circuits combinatoires ; le troisième porte sur l'algorithmique.

Des questions de programmation parsèment l'énoncé et vous donneront l'occasion de faire la preuve de votre parfaite maîtrise du langage Caml.

Les questions marquées ⋆⋆ sont plus difficiles.

Veuillez rédiger chaque partie sur une copie séparée.

Table des matières

1 Logique

▶ On dispose d'un ensemble $\mathcal{V} = \{x, y, z, \dots\}$ de *variables logiques*. Un *littéral* est une variable x ou sa négation $\neg x$. Une *clause* est un littéral ou une disjonction de littéraux : $x \vee z$, $\neg y$ et $(\neg x) \vee y \vee z$ sont des clauses.

▶ Une *assignation* est une fonction φ de $\mathcal{V}$ dans l'ensemble $\{\text{vrai}, \text{faux}\}$. Les tableaux suivants permettent d'étendre φ aux littéraux, puis aux clauses :

$\varphi(x)$	$\varphi(\neg x)$
faux	vrai
vrai	faux

$\varphi(c)$	$\varphi(c')$	$\varphi(c \vee c')$
faux	faux	faux
faux	vrai	vrai
vrai	faux	vrai
vrai	vrai	vrai

▶ Une assignation φ *satisfait* une clause c si $\varphi(c) = \text{vrai}$.

Question 1 Énumérez les assignations qui satisfont simultanément toutes les clauses de l'ensemble E suivant :

$$E = \big\{x \vee (\neg y) \vee z, (\neg x) \vee (\neg y) \vee z, (\neg x) \vee y \vee (\neg z), x \vee y \vee (\neg z), (\neg x) \vee (\neg y) \vee (\neg z)\big\}$$

Question 2 Déterminez le nombre maximal de clauses que l'on peut satisfaire simultanément, dans l'ensemble F suivant :

$$F = \{x, y, z, w, (\neg x) \vee (\neg y), (\neg y) \vee (\neg z), (\neg z) \vee (\neg x), x \vee (\neg w), y \vee (\neg w), z \vee (\neg w)\}$$

2 Circuits combinatoires

▶ Parmi les instructions dont dispose un ordinateur, on trouve `INR` (incrémentation) : cette instruction permet d'ajouter 1 à une case mémoire. Si cet ordinateur possède une architecture n bits, l'addition est effectuée modulo 2^n. On se propose d'étudier un circuit logique effectuant cette opération, dans les cas $n = 1$ et $n = 2$.

▶ On note $\mathcal{B} = \{0,1\}$. Cet ensemble est muni des deux lois $+$ et $\cdot$ décrites par les tables ci-dessous :

$+$	0	1
0	0	1
1	1	1

$\cdot$	0	1
0	0	0
1	0	1

Si $x \in \mathcal{B}$, on note $\overline{x} = 1 - x$.

▶ Les portes logiques *élémentaires* sont la porte ET à deux entrées, la porte OU à deux entrées, et la porte NON (à une entrée) ; elles sont décrites dans la figure 1.

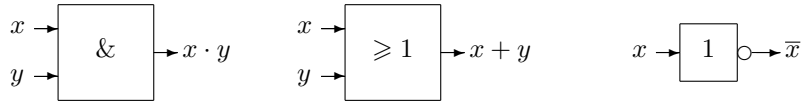

Figure 1: les trois portes logiques élémentaires

Question 1 Lorsque $n = 1$, quel circuit très simple réalise l'incrémentation ?

▶ Dans le cas $n = 2$, le circuit logique qui nous intéresse possède deux entrées e_0 et e_1, et deux sorties s_0 et s_1. La fonction de ce circuit est parfaitement définie par la relation :

$$s_0 + 2s_1 \equiv e_0 + 2e_1 + 1 \pmod 4$$

Question 2 Dressez une table donnant les valeurs de s_0 et s_1 pour chacun des quatre cas de figure possibles.

Question 3 Donnez des formules exprimant s_0 et s_1 en fonction de e_0 et e_1.

Question 4 Dessinez le circuit logique d'incrémentation ; vous placerez les entrées sur le côté gauche de la figure, et les sorties sur le côté droit.

3 Algorithmique : le problème de partition

3.1 Présentation du problème

▶ Après un casse réussi, deux cambrioleurs veulent partager équitablement leur butin. Ils commencent par s'accorder sur la valeur de chacun des n objets volés ; ceci leur donne un ensemble $X = \{x_0, \ldots, x_{n-1}\}$ de nombres entiers[1]. Ils leur faut alors résoudre le problème suivant : déterminer une partition de X en deux sous-ensembles Y et Z de même somme.

▶ Nos deux malfaiteurs sont confrontés au *problème de partition* : nous dirons que X est une *instance* de ce problème. Le *problème de décision* associé s'énonce ainsi : une telle partition «équilibrée» existe-t-elle ?

▶ Nous noterons $|X|$ le cardinal de X et $\|X\|$ la somme des éléments de X : $\|X\| = \sum_{x \in X} x$.

▶ **Attention** : l'emploi de références est interdit. Dans la question 11, on demande de construire une structure de données mutable (matrice) ; vous ne devez modifier chaque membre d'une telle structure qu'une fois au plus.

3.2 Résolution à l'aide d'un oracle

▶ Un *oracle* est un respectable personnage qui, si nous lui présentons X et un entier k (ainsi qu'un modeste cadeau[2]), condescend à nous dire s'il existe un sous-ensemble Y de X tel que $\|Y\| = k$. Notez bien que l'oracle ne nous donne pas un tel sous-ensemble : il se contente de résoudre un problème de décision. Si $2k = \|X\|$, il s'agit précisément du celui qui est associé au problème de partition.

Question 1 • Expliquez comment résoudre le problème de partition au prix de $n + \mathcal{O}(1)$ consultations de l'oracle.

▶ Considérons la fonction `toto` suivante :

```
let rec toto s = function
 | []  -> s = 0
 | t::q -> (s = t) or ((s > t) & toto (s-t) q) or toto s q ;;
```

Question 2 • Quel est le type de `toto` ?

Question 3 • Expliquez ce que calcule la fonction `toto`.

Question 4 • Utilisez `toto` pour rédiger en Caml une fonction de signature

```
partition : int list -> int list
```

spécifiée comme suit :

- si le problème de partition pour X possède une solution, alors `partition x` rendra une liste extraite de `x`, et dont la somme sera exactement la moitié de $\|X\|$;
- sinon, une exception sera levée.

▶ Hélas, de nos jours les oracles ne sont pas légion. Nous pourrions nous rabattre sur la méthode de la force brutale : elle consiste à énumérer toutes les parties de X et à calculer la somme de chacune d'elles, dans l'espoir d'en trouver une dont la somme soit exactement la moitié de $\|X\|$.

Question 5 • Quelle remarque *très simple* permet d'économiser exactement *la moitié* des calculs ?

[1] Curieux, ces cambrioleurs qui numérotent à partir de zéro ; s'agirait-il d'informaticiens reconvertis ?
[2] Un flacon de *Cardhu*, par exemple

3.3 Résolution par programmation dynamique

▶ Mauvaise nouvelle : le problème de partition est NP-complet. Il n'y a donc pas d'espoir *raisonnable* de trouver un algorithme de coût polynomial pour le résoudre.

▶ Bonne nouvelle : il existe une solution du problème de partition par programmation dynamique, de coût $\mathcal{O}(n\|X\|)$, où $n = |X|$. Nous allons l'examiner de ce pas !

▶ Associons à l'instance X du problème une matrice de booléens m à n lignes (indexées de 0 à $n-1$) et $\|X\|+1$ colonnes (indexées de 0 à $\|X\|$). Cette matrice est définie comme suit :

- $m(i,j) =$ `vrai` s'il existe une partie Y de $\{x_0, \ldots, x_i\}$ vérifiant $\|Y\| = j$;
- $m(i,j) =$ `faux` dans le cas contraire.

Question 6 • Comment la ligne 0 de la matrice doit-elle être remplie ?

Question 7 • Montrez que l'on peut remplir la $(i+1)$-ième ligne de m dès que l'on connaît la i-ième.

Question 8 • Expliquez où se trouve la réponse au problème de décision, dans la matrice m.

Question 9 ⋆⋆ • Expliquez comment exploiter la matrice m pour obtenir la réponse au problème de partition.

Question 10 • Construisez la matrice m associée à $X = \{5, 2, 3\}$.

Question 11 • Rédigez en Caml une fonction de signature

```
partition_dynamique : int list -> bool vect vect
```

spécifiée comme suit : `partition_dynamique x` construit la matrice m associée à l'instance X du problème de partition.

Question 12 • Peut-on, comme à la question 5, faire l'économie de certains calculs ?

3.4 Annexe : boîte à outils Caml

▶ Voici la description de la boîte à outils dont vous pouvez disposer. Tout d'abord, les signatures des différentes fonctions :

```
vect_of_list : 'a list -> 'a vect
sum : int list -> int
make_matrix : int -> int -> 'a -> 'a vect vect
```

Voici maintenant la spécification de chacune de ces fonctions :

- `vect_of_list` convertit une liste en un vecteur ; par exemple, `vect_of_list [9;2;3]` rendra le vecteur `[|9;2;3|]`.
- `sum l` calcule la somme des éléments de la liste ℓ ; par exemple, `sum [7;18;-3;5]` rendra `27`.
- `make_matrix n p x` construit une matrice à n lignes et p colonnes, dont tous les coefficients ont pour valeur x.

FIN