Architecture des ordinateurs : TD4

Université de Tours

`Département informatique de Blois`

La mémoire

*
* *

Problème 1

1. Soit la table de vérité suivante :

J	K	Q_{n+1}	$\overline{Q}_{n+1}$
0	0	Q_n	$\neg Q_n$
0	1	0	1
1	0	1	0
1	1	$\neg Q_n$	Q_n

2. En représentant une entrée par un code $\langle JK \rangle \in \{00, 01, 10, 11\}$, dessiner l'automate de Moore de la sortie Q.

3. Équation logique :

   (a) Dresser la table de Karnaugh de la table de vérité ci-dessus et montrer que $Q_{n+1} = (J \wedge \neg Q_n) \vee (\neg K \wedge Q_n)$

   (b) Donner l'équation booléenne de Q_{n+1} uniquement à l'aide de connecteur $\uparrow$.

4. Dessiner le circuit séquentielle correspondant à table de vérité ci-dessus, on dessinera le circuit synchrone avec une entrée horloge ϕ.

Problème 2

1. Soit un mot $x = x_1...x_n \in \{0,1\}^n$, on considère une méthode φ de codage par checksum.

   (a) Montrer que $\forall x \in C_\varphi, w(x) = 0 \mod 2$.

   (b) Écrire un algorithme permettant d'ajouter à un `char` sa clé de parité.

2. Soit un mot $x = x_1...x_n \in \{0,1\}^n$. On considère une application de codage φ par répétition pure, c'est-à-dire un code (nk, n) où le mot x est simplement répété k fois d'affilé.

   (a) Quelle est la distance minimale δ entre deux mots de code par répétition pure ?

   (b) Proposer un code de répétition qui soit 2-correcteur, quel est son rendement ?

   (c) Proposer un algorithme d'encodage par répétition de `char` qui soit 1-correcteur.

Problème 3

On rappelle que les caractères ASCII sont contenus dans des octets mais ne sont codés que sur 7 bits afin que l'on puisse leur adjoindre un bit de parité. De même, si l'on groupe plusieurs caractères ensemble, il est possible d'appliquer une méthode de codage φ par *parité longitudinale et transversale*.
Ainsi, pour un texte $x = x_1...x_n \in \text{ASCII}^n$ de n caractères, on associe X, la forme matricielle de x de taille $n \times 7$ où chaque ligne i correspond au codage binaire du caractère ASCII x_i.
On donne en annexe la table de codage des caractères ASCII.
Par exemple, le texte $x =$ "`chat`" possède la matrice associée :

$$X = \begin{pmatrix} \langle\text{'c'}\rangle_2 \\ \langle\text{'h'}\rangle_2 \\ \langle\text{'a'}\rangle_2 \\ \langle\text{'t'}\rangle_2 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

On note $\varphi(X)$ la matrice de codage par parité longitudinale et transversale résultante de X. Sa taille est de $(n+1) \times 8$.

1. Montrer que si la matrice de codage $\varphi(X)$ comporte un nombre impair de bits à 1, alors une erreur est détectée.

2. Soit le texte $x =$ "`texte x`". Déterminer la matrice $\varphi(X)$ associée à x.

3. Écrire un programme `int[] encodage(String x)` en Java / C qui encode une chaîne de caractères ASCII selon la méthode par parité longitudinale et transversale et redonne un tableau d'entiers `tab` où `tab[i]` correspond au codage décimal de la ligne binaire i de $\varphi(X)$.

   Ex : L'appel `encodage("chat")` retourne le tableau $[198, 209, 195, 232, 60]$.

4. Soit un texte y stocké en mémoire. On donne la matrice $\varphi(Y) = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \end{pmatrix}$.

   Que pouvez-vous déduire de $\varphi(Y)$? Décoder le texte y correspondant, éventuellement corrigé.

Problème 4

Une matrice est conceptuellement un tableau de nombres à 2 dimensions, mais en interne un tableau ne peut avoir qu'une seule dimension. Pour cela, il y a 2 techniques :

- Pour la 1ère dimension (par exemple n° de ligne) on stocke un tableau d'adresses. Chaque adresse renvoie au tableau des éléments de la ligne (éléments numérotés par leur colonne). On a donc un tableau d'adresses de tableaux de nombres. Technique Java, Caml, Python... quand on écrit `M[i][j]`.

- On stocke tous les éléments de la matrice dans un seul grand tableau `t`, et l'élément M_{ij} d'une matrice s'obtient par `M[formule en i et j]`. Technique du C et du Fortran, moins souple, plus efficace.

Cette dernière technique admet 2 variantes, selon le sens dans lequel on représente les données. Voici par exemple la matrice :

$$M = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

- Convention du langage C : les éléments d'une même ligne sont toujours consécutifs, et on stocke les lignes à la suite (donc les données sont stockées dans l'ordre habituel de l'écriture). Exemple : M se stocke $(abcdefghi)$
- Convention du Fortran : les éléments d'une même colonne sont toujours consécutifs, et on stocke les colonnes à la suite. Exemple : M se stocke $(adgbehcfi)$.

On adopte les conventions du C.

1. Soit M une matrice à n lignes et p colonnes, dont les indices commencent à 0 (càd : $0 \leq i < n$ et $0 \leq j < p$). Donner la formule `M[...]` qui fournit M_{ij}.
2. Soient A une matrice à n lignes et q colonnes et B une matrice à q lignes et p colonnes, on note $M = AB$. Pour calculer M, on calcule chaque produit contractant

   $$M_{ij} = \sum_{k=0}^{q-1} A_{ik} B_{kj}$$

   Écrire un algorithme en Java ou en C qui multiplie deux matrices sous forme de tableau à une dimension selon la convention du C.
3. Dans chaque produit contractant de l'algorithme précédent, que dire des accès mémoire concernant A ? Concernant B ?
4. Au lieu de calculer AB, quel calcul similaire à AB faudrait-il effectuer pour accélérer les accès mémoire en lecture ?
5. Écrire un algorithme qui calcule la transposée d'une matrice, puis un algorithme qui multiplie deux matrices avec le maximum d'efficacité mémoire en s'appuyant sur les principes de localité des caches.

Annexe – Table ASCII

Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex
(nul)	0	0000	0x00	(sp)	32	0040	0x20	@	64	0100	0x40	‘	96	0140	0x60
(soh)	1	0001	0x01	!	33	0041	0x21	A	65	0101	0x41	a	97	0141	0x61
(stx)	2	0002	0x02	"	34	0042	0x22	B	66	0102	0x42	b	98	0142	0x62
(etx)	3	0003	0x03	#	35	0043	0x23	C	67	0103	0x43	c	99	0143	0x63
(eot)	4	0004	0x04	$	36	0044	0x24	D	68	0104	0x44	d	100	0144	0x64
(enq)	5	0005	0x05	%	37	0045	0x25	E	69	0105	0x45	e	101	0145	0x65
(ack)	6	0006	0x06	&	38	0046	0x26	F	70	0106	0x46	f	102	0146	0x66
(bel)	7	0007	0x07	’	39	0047	0x27	G	71	0107	0x47	g	103	0147	0x67
(bs)	8	0010	0x08	(	40	0050	0x28	H	72	0110	0x48	h	104	0150	0x68
(ht)	9	0011	0x09	)	41	0051	0x29	I	73	0111	0x49	i	105	0151	0x69
(nl)	10	0012	0x0a	*	42	0052	0x2a	J	74	0112	0x4a	j	106	0152	0x6a
(vt)	11	0013	0x0b	+	43	0053	0x2b	K	75	0113	0x4b	k	107	0153	0x6b
(np)	12	0014	0x0c	,	44	0054	0x2c	L	76	0114	0x4c	l	108	0154	0x6c
(cr)	13	0015	0x0d	-	45	0055	0x2d	M	77	0115	0x4d	m	109	0155	0x6d
(so)	14	0016	0x0e	.	46	0056	0x2e	N	78	0116	0x4e	n	110	0156	0x6e
(si)	15	0017	0x0f	/	47	0057	0x2f	O	79	0117	0x4f	o	111	0157	0x6f
(dle)	16	0020	0x10	0	48	0060	0x30	P	80	0120	0x50	p	112	0160	0x70
(dc1)	17	0021	0x11	1	49	0061	0x31	Q	81	0121	0x51	q	113	0161	0x71
(dc2)	18	0022	0x12	2	50	0062	0x32	R	82	0122	0x52	r	114	0162	0x72
(dc3)	19	0023	0x13	3	51	0063	0x33	S	83	0123	0x53	s	115	0163	0x73
(dc4)	20	0024	0x14	4	52	0064	0x34	T	84	0124	0x54	t	116	0164	0x74
(nak)	21	0025	0x15	5	53	0065	0x35	U	85	0125	0x55	u	117	0165	0x75
(syn)	22	0026	0x16	6	54	0066	0x36	V	86	0126	0x56	v	118	0166	0x76
(etb)	23	0027	0x17	7	55	0067	0x37	W	87	0127	0x57	w	119	0167	0x77
(can)	24	0030	0x18	8	56	0070	0x38	X	88	0130	0x58	x	120	0170	0x78
(em)	25	0031	0x19	9	57	0071	0x39	Y	89	0131	0x59	y	121	0171	0x79
(sub)	26	0032	0x1a	:	58	0072	0x3a	Z	90	0132	0x5a	z	122	0172	0x7a
(esc)	27	0033	0x1b	;	59	0073	0x3b	[	91	0133	0x5b	{	123	0173	0x7b
(fs)	28	0034	0x1c	<	60	0074	0x3c	\	92	0134	0x5c	|	124	0174	0x7c
(gs)	29	0035	0x1d	=	61	0075	0x3d	]	93	0135	0x5d	}	125	0175	0x7d
(rs)	30	0036	0x1e	>	62	0076	0x3e	^	94	0136	0x5e	~	126	0176	0x7e
(us)	31	0037	0x1f	?	63	0077	0x3f	_	95	0137	0x5f	(del)	127	0177	0x7f