A 2001 – INF – MP

ÉCOLE NATIONALE DES PONTS ET CHAUSSÉES,
ÉCOLES NATIONALES SUPÉRIEURES DE L'AÉRONAUTIQUE ET DE L'ESPACE,
DES TECHNIQUES AVANCÉES, DES TÉLÉCOMMUNICATIONS,
DES MINES DE PARIS, DES MINES DE SAINT-ÉTIENNE, DES MINES DE NANCY,
DES TÉLÉCOMMUNICATIONS DE BRETAGNE,
ÉCOLE POLYTECHNIQUE
(Filière T.S.I.)

CONCOURS D'ADMISSION 2001

ÉPREUVE D'INFORMATIQUE

Filière MP
(Durée de l'épreuve : 3 heures)

Sujet mis à la disposition des concours Cycle International, ENSTIM et TPE-EIVP.

Les candidats et les candidates sont priés de mentionner de façon apparente sur la première page de la copie :
« INFORMATIQUE - Filière MP »

RECOMMANDATIONS AUX CANDIDATS ET CANDIDATES

- **L'énoncé de cette épreuve, y compris cette page de garde, comporte 8 pages.**
- **Si, au cours de l'épreuve, un candidat ou une candidate repère ce qui lui semble être une erreur d'énoncé, il ou elle le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il ou elle a décidé de prendre.**
- **Tout résultat fourni dans l'énoncé peut être utilisé pour les questions ultérieures même s'il n'a pas été démontré.**
- **Il ne faut pas hésiter à formuler les commentaires qui semblent pertinents même lorsque l'énoncé ne les demande pas explicitement.**
- **L'utilisation d'une calculatrice ou d'un ordinateur est interdite.**

COMPOSITION DE L'ÉPREUVE

L'épreuve comprend trois problèmes indépendants :

- le problème de théorie des automates, n° 1 page 2, à résoudre en 1 h 15 min environ ;
- le problème de complexité algorithmique, n° 2 page 5, à résoudre en 60 min environ ;
- le problème de logique des propositions n° 3 page 7, à résoudre en 45 min environ.

Attention ! Dans chacun des deux premiers problèmes, les candidats et les candidates sont invités à donner le nom du langage de programmation, Pascal ou Caml, dans lequel les algorithmes seront écrits lorsque des questions le demanderont. Seuls les programmes écrits dans le langage choisi seront corrigés.

1 Problème sur les automates finis — 1 h 15 min environ

Rappels pour fixer les notations et la terminologie

Un *alphabet* A est un ensemble fini d'éléments appelés *lettres*. A^* est l'ensemble des suites finies de lettres de A, appelées *mots*. Un *langage* est une partie de A^*.

Un *automate* $\mathcal{A}$ est un *graphe orienté et étiqueté*, décrit par une structure $<Q,A,E,I,T>$:

- A est un alphabet ;
- Q est un ensemble fini, non vide et appelé ensemble des *états* ; ce sont les *sommets* du graphe ;
- $E \subseteq Q \times A \times Q$ est appelé l'ensemble des *transitions*, ce sont les *arcs* du graphe ;
- $I \subseteq Q$ est appelé ensemble des *états initiaux* de l'automate ;
- $T \subseteq Q$ est appelé ensemble des *états terminaux* de l'automate.

Une transition $(p,a,q) \in E$ est un arc du graphe, d'étiquette a et allant de l'état p vers l'état q que l'on pourra noter $p \xrightarrow{a} q$.

La représentation graphique d'un automate obéit aux conventions suivantes :

- un état e est figuré par un cercle marqué en son centre par e ; si $e \in I$, cela est figuré par une flèche entrante sans origine ; si $e \in T$, cela est figuré par une flèche sortante sans but ;
- une transition $(p,a,q) \in E$ est figurée par une flèche allant de l'état p vers l'état q ; cette flèche est étiquetée par la lettre a.

Exemple : une telle représentation graphique est illustrée dans la question 3 page ci-contre.

Un *calcul* de $\mathcal{A}$ est un chemin $p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} p_2 \cdots \xrightarrow{a_n} p_n$ dans le graphe : pour $i \in [1,n]$ on a $p_{i-1} \xrightarrow{a_i} p_i \in E$. On dit que p_0 est l'*origine* du calcul, p_n est son *extrémité*, et que le calcul *traverse* les états $p_0, p_1, \ldots, p_n$.

L'*étiquette* du calcul est le mot formé par la suite des étiquettes des arcs successifs du chemin. Un calcul d'origine p, d'extrémité q et d'étiquette u peut être noté $p \xrightarrow{u} q$. Il est dit *réussi* lorsque p est un état initial et q est un état terminal. Un mot de $u \in A^*$ est *reconnu* par $\mathcal{A}$ s'il existe un calcul réussi de $\mathcal{A}$ d'étiquette u. $L(\mathcal{A})$, le *langage reconnu* par $\mathcal{A}$, est l'ensemble des étiquettes des calculs réussis.

L'automate $\mathcal{A}$ est dit *émondé* si pour chaque état p de $\mathcal{A}$, il existe un état initial i, un état terminal t et deux mots u et v tels que $i \xrightarrow{u} p$ et $p \xrightarrow{v} t$ sont des calculs, c'est-à-dire que tout état est réellement utile à la reconnaissance des mots de $L(\mathcal{A})$.

Problème

Dans ce problème, on se donne un automate $\mathcal{A} = <Q,A,E,I,T>$. Nous allons écrire un algorithme permettant de l'émonder, c'est-à-dire de déterminer ses états inutiles.

1 – Il faudra rédiger des algorithmes. Choisir le langage, Pascal ou Caml, dans lequel seront rédigés ces algorithmes. **Attention !** Tous les algorithmes demandés dans ce problème devront être écrits dans le langage choisi.

Représentation informatique d'un automate. Soit N_E et N_L des constantes entières strictement positives. On suppose que l'automate a N_E états numérotés de 1 à N_E. Pour $i \in [1,N_E]$, on notera e_i l'état numéro i. On suppose que l'alphabet contient N_L lettres numérotées de 1 à N_L. Pour $j \in [1,N_L]$, on notera l_j la lettre numéro j. On choisit de représenter l'automate ainsi :

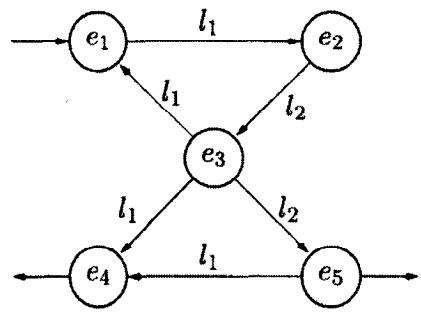
- à un état e_i et une lettre l_j, on associe un vecteur de N_E booléens, noté TL_i^j, tel que pour tout $k \in [1,N_E]$, $TL_i^j[k]$ est *vrai* si et seulement si $(e_i,l_j,e_k) \in E$; le vecteur TL_i^j est une description des transitions issues de e_i et étiquetées par l_j ;
- l'ensemble des transitions issues d'un état e_i est représenté par le vecteur noté TR_i des N_L vecteurs TL_i^j pour $1 \leqslant j \leqslant N_L$; c'est-à-dire $TR_i[j] = TL_i^j$ pour tout $j \in [1,N_L]$;
- le graphe orienté et étiqueté de l'automate est représenté par le vecteur G des N_E vecteurs TR_i pour $1 \leqslant i \leqslant N_E$; c'est-à-dire $G[i] = TR_i$ pour tout $i \in [1,N_E]$.

On remarque que l'on ne représente que les transitions et que l'on ne représente pas encore le caractère initial ou terminal des états. Cela sera pris en compte plus tard dans le problème.

Important ! Dans les calculs de complexité qui seront demandés, on admettra que les opérations élémentaires du langage de programmation (opérations arithmétiques, comparaisons arithmétiques, accès et modifications de la valeur d'une variable ou d'un élément de tableau, etc.) ont une complexité en $O(1)$.

2 – Dans le langage choisi, décrire comment représenter, en accord avec les éléments qui viennent d'être donnés, le graphe orienté et étiqueté d'un automate.

3 – Toujours dans le langage choisi et en supposant $N_E = 5$ et $N_L = 2$, utiliser la réponse à la question précédente pour représenter l'automate suivant :



Émonder l'automate revient à déterminer ses états inutiles. Un état e est *inutile* s'il n'existe pas de calcul dont l'origine est un état initial et dont l'extrémité est e ou bien s'il n'existe pas de calcul dont l'origine est e et dont l'extrémité est un état terminal.

Matrice d'accesibilité d'un automate. Pour déterminer les états inutiles, il faudra calculer la matrice d'accessibilité de l'automate. Cette matrice est une matrice de booléens C, carrée de dimension N_E, telle que pour $(i,j) \in [1,N_E] \times [1,N_E]$, $C[i,j]$ vaut *vrai* si et seulement s'il existe un calcul d'origine e_i et d'extrémité e_j.

4 – Dans le langage choisi, décrire comment représenter la matrice d'accessibilité d'un automate.

L'algorithme de Roy-Warshall. L'algorithme qui va être utilisé s'appelle algorithme de Roy-Warshall. Cet algorithme commence par initialiser la matrice C : pour $(i,j) \in [1,N_E] \times [1,N_E]$, $C[i,j]$ est initialisé à *vrai* si et seulement si $i = j$ ou bien s'il existe une transition de e_i vers e_j dans l'automate. L'algorithme transforme la matrice C initialisée, que l'on nommera C_0, en N_E étapes. Pour $k \in [1,N_E]$, on note C_k la matrice à l'issue de k-ième étape de transformation. L'idée de cet algorithme est que pour tous $k \in [0,N_E]$ et $(i,j) \in [1,N_E] \times [1,N_E]$, $C_k[i,j]$ est *vrai* si et seulement si $i = j$ ou bien s'il existe un calcul d'origine e_i, d'extrémité e_j et tel qu'aucun état traversé par ce calcul à l'exception de l'origine et de l'extrémité n'a un numéro strictement supérieur à k. On dit alors que la matrice C_k vérifie *la propriété au rank* k.

5 – Dans le langage choisi, écrire un algorithme permettant d'initialiser la matrice d'accessibilité C à partir de la représentation G de l'automate.

6 – Quelle est la complexité en temps de l'algorithme de la question 5 en fonction de N_E et N_L ?

7 – Vérifier trivialement que C_0 vérifie la propriété au rang 0.

8 – Vérifier trivialement que si C_{N_E} vérifie la propriété au rang N_E alors C_{N_E} est la matrice d'accessibilité du graphe.

9 – Soit $k \in [0,N_E-1]$, on suppose C_k vérifie la propriété au rang k et C_{k+1} vérifie la propriété au rang $k+1$. Soit $(i,j) \in [1,N_E] \times [1,N_E]$, démontrer que $C_{k+1}[i,j]$ est *vrai* si et seulement si $C_k[i,j]$ est *vrai* ou bien si $C_k[i,k+1]$ et $C_k[k+1,j]$ sont *vrais*.

10 – Soit $k \in [0,N_E-1]$, on suppose C_k vérifie la propriété au rang k et C_{k+1} vérifie la propriété au rang $k+1$. Soit $(i,j) \in [1,N_E] \times [1,N_E]$, démontrer que :

a – $C_k[i,k+1]$ est *vrai* si et seulement si $C_{k+1}[i,k+1]$ est *vrai* ;

b – $C_k[k+1,j]$ est *vrai* si et seulement si $C_{k+1}[k+1,j]$ est *vrai*.

11 – Déduire et justifier à l'aide des questions 9 et 10 un algorithme permettant de transformer la matrice C_k en C_{k+1} pour tout $k \in [0,N_E - 1]$.

12 – Quelle est la complexité en temps de l'algorithme de la question 11 en fonction de N_E et N_L ?

13 – Dans le langage de programmation choisi, écrire un algorithme prenant en argument la représentation G de l'automate et produisant sa matrice C d'accessibilité.

14 – Quel est la complexité en temps de l'algorithme de la question 13 en fonction de N_E et N_L ?

On suppose que les états initiaux sont décrits par un vecteur I de N_E booléens tel que pour tout $k \in [1,N_E]$, $I[k]$ est *vrai* si et seulement si e_k est un état initial. On suppose également que les états terminaux sont décrits par un vecteur F de N_E booléens tel que pour tout $k \in [1,N_E]$, $F[k]$ est *vrai* si et seulement si e_k est un état terminal. On désire obtenir le vecteur U de N_E booléens tel que pour tout $k \in [1,N_E]$, $U[k]$ est *vrai* si et seulement si e_k est un état inutile.

15 – Dans le langage de programmation choisi, décrire comment représenter les vecteurs I, F et U.

16 – Toujours dans le langage de programmation choisi, écrire un algorithme prenant en arguments la représentation G de l'automate, les représentations des vecteurs I et F décrivant ses états initiaux et terminaux et produisant la représentation du vecteur U de ses états inutiles.

17 – Quel est la complexité en temps de l'algorithme de la question 16 en fonction de N_E et N_L ?

18 – L'hypothèse a été faite en début de problème que les opérations élémentaires du langage de programmation ont une complexité en $O(1)$. Expliquer pourquoi cette hypothèse est réaliste pour le langage de programmation choisi.

FIN DU PROBLÈME SUR LES AUTOMATES

2 Problème de complexité algorithmique — 60 min environ

Calcul efficace de x^n

Le problème du calcul efficace de x^n pour n entier positif devient crucial lorsque n est particulièrement grand ou bien lorsque x est un type de données où le produit est algorithmiquement coûteux. C'est le cas si x est une matrice carrée de grande dimension ou bien si x est un polynôme de degré élevé. Dans ces cas, le temps d'exécution d'une multiplication n'est plus négligeable et il peut être raisonnable de rechercher des algorithmes où l'on cherche à minimiser le nombre de multiplications effectuées.

Ce qui intéresse l'informaticien dans le calcul de x^n est le nombre de multiplications par x où par des puissances déjà calculées de x. Pour essayer de compter et de minimiser ce nombre de multiplications, on peut se contenter d'étudier le cas où x est un entier et n un entier strictement positif. C'est ainsi que la question est abordée dans ce problème.

Terminologie et notation. Si $n > 0$ est un entier :

- on appelle *représentation binaire minimale* (en abrégé *r.b.m.*) de n, la représentation binaire de n où l'on ne considère que les chiffres significatifs, c'est-à-dire que l'on élimine tous les éventuels chiffres 0 figurant en tête (à gauche) de la représentation ;
- on note $\lambda(n)$ le nombre de chiffres de la représentation binaire minimale de n ;
- on note $\nu(n)$ le nombre de chiffres égaux à 1 dans la représentation binaire minimale de n.

1 – Il faudra rédiger des algorithmes. Choisir le langage, Pascal ou Caml, dans lequel seront rédigés ces algorithmes. **Attention !** Tous les algorithmes demandés dans ce problème devront être écrits dans le langage choisi.

La méthode élémentaire

2 – L'algorithme le plus simple pour calculer x^n est basé sur les équations suivantes :

$$\begin{cases} x^1 &= x \\ x^{k+1} &= x\,x^k \qquad \text{si } k > 0 \end{cases}$$

a – Dans le langage choisi, écrire un algorithme *récursif*, nommé es, prenant en arguments un entier x et un entier $n > 0$ et calculant x^n en utilisant les équations ci-dessus.

b – Combien de multiplications sont-elles effectuées par l'algorithme es en fonction de son deuxième argument n ?

La méthode de Legendre

3 – L'algorithme de Legendre considère la représentation binaire minimale de l'exposant entier $n > 0$. L'algorithme pour calculer x^n s'énonce ainsi :

- on part de la valeur 1 ;
- on parcourt la r.b.m. de n de gauche à droite ; pour chaque chiffre rencontré, on élève la valeur au carré, puis si le chiffre est 1, on multiplie la valeur par x.

a – Démontrer que cet algorithme est correct. Attention ! Il est inutile de le programmer.

b – Exprimer le nombre de multiplications qui sont effectuées par cet algorithme en fonction de $\lambda(n)$ et $\nu(n)$ où n est l'exposant ?

La méthode dichotomique

4 – L'algorithme *dichotomique* pour calculer x^n est basé sur les équations suivantes :

$$\begin{cases} x^1 & = & x & \\ x^{2k} & = & (x^2)^k & \text{si } k > 0 \\ x^{2k+1} & = & x\,x^{2k} & \text{si } k > 0 \end{cases}$$

On suppose que le langage de programmation choisi dispose d'une fonction *pair* dont l'argument est un entier et le résultat est un booléen qui est *vrai* si et seulement si l'argument est pair.

a – Dans le langage choisi, écrire un algorithme *récursif*, nommé *ed*, prenant en arguments un entier x et un entier $n > 0$ et calculant x^n en utilisant les équations ci-dessus.

b – En examinant l'algorithme *ed*, écrire un algorithme *récursif*, nommé *cd*, prenant en argument un entier $n > 0$ et calculant le nombre de multiplications effectuées pendant le calcul de $ed(x,n)$ pour un x entier quelconque.

c – Soit $k > 0$ un entier, exprimer $\lambda(2k)$ en fonction de $\lambda(k)$ et exprimer $\lambda(2k+1)$ en fonction de $\lambda(2k)$. En déduire et écrire un algorithme *récursif*, nommé *lambda*, prenant en argument un entier $n > 0$ et calculant $\lambda(n)$.

d – Soit $k > 0$ un entier, exprimer $\nu(2k)$ en fonction de $\nu(k)$ et exprimer $\nu(2k+1)$ en fonction de $\nu(2k)$. En déduire et écrire un algorithme *récursif*, nommé *nu*, prenant en argument un entier $n > 0$ et calculant $\nu(n)$.

e – Pour $n > 0$ entier, on pose $\pi(n) = \lambda(n) + \nu(n) - 2$. En utilisant les résultats des deux questions précédentes, déduire et écrire un algorithme *récursif*, nommé *pi*, prenant en argument un entier $n > 0$ et calculant $\pi(n)$.

f – Quel est le nombre de multiplications effectuées par l'algorithme *ed* en fonction de $\lambda(n)$ et $\nu(n)$ où n, entier strictement positif, est le deuxième argument ?

g – Quel est le nombre de multiplications effectuées par l'algorithme *ed* en fonction de son deuxième argument n lorsque celui-ci est de la forme 2^k où $k \geqslant 0$ est un entier naturel.

La méthode des facteurs

L'algorithme dichotomique réduit sensiblement le nombre de multiplications par rapport à l'algorithme le plus simple de la question 2 page précédente. Mais il n'est pas optimal. On a $\pi(15) = 6$ mais en remarquant que $15 = 3 \times 5$ et donc $x^{15} = (x^3)^5$, on peut calculer x^{15} en 5 multiplications : on calcule x^3 en 2 multiplications à l'aide de la méthode dichotomique puis on l'élève à la puissance 5 en 3 multiplications toujours à l'aide de la méthode dichotomique. Cela donne :

$$x^2 = x\,x \qquad x^3 = x\,x^2 \qquad x^6 = x^3\,x^3 \qquad x^{12} = x^6\,x^6 \qquad x^{15} = x^{12}\,x^3$$

Cette méthode, appelée *méthode des facteurs*, utilise le fait que si $n = p\,q$ alors on a $x^n = (x^p)^q$.

5 – En remarquant que $33 = 3 \times 11$, appliquer la méthode des facteurs esquissée ci-dessus et vérifier qu'elle n'est pas optimale.

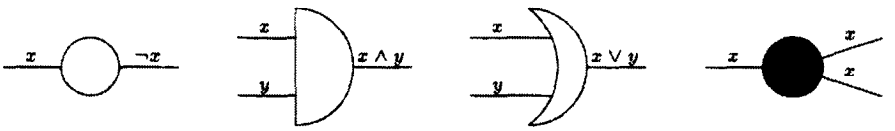
FIN DU PROBLÈME DE COMPLEXITÉ ALGORITHMIQUE

3 Problème de logique des propositions — 45 min environ

Afin de construire des circuits logiques, on se donne les trois portes logiques élémentaires correspondant aux trois connecteurs suivants :

- le connecteur unaire et préfixé de la négation noté $\neg$;
- le connecteur binaire et infixé de la conjonction noté $\wedge$;
- le connecteur binaire et infixé de la disjonction noté $\vee$.

On se donne également le *nœud duplicateur* produisant sur ses deux sorties la valeur fournie sur son unique entrée. Ces éléments de circuits logiques sont représentés graphiquement ainsi :



Porte NON — Porte ET — Porte OU — Nœud duplicateur

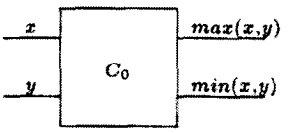
N.B. Les circuits logiques construits en réponse à une question quelconque de ce problème devront être construits en n'utilisant que les élements de circuits logiques donnés ci-dessus et ceux construits en réponses aux questions qui précédent cette question.

Si x et y sont deux valeurs numériques, on note $max(x,y)$ la plus grande de ces deux valeurs et $min(x,y)$ la plus petite de ces deux valeurs.

Soit n un entier strictement positif et x un entier tel que $0 \leqslant x < 2^n$, on note $\overline{x_{n-1} \cdots x_1 x_0}^2$ la représentation en base 2 sur n bits de x qui est définie comme suit : on prend la représentation en base 2 de x qui est de longueur au plus n et on la complète éventuellement avec des zéros à gauche pour que sa longueur soit exactement n.

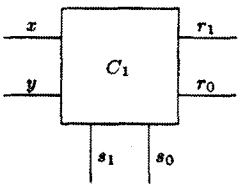
Un comparateur logique

1 – Construire un circuit logique C_0 prenant en entrées deux bits d'information x et y et fournissant en sorties les valeurs $max(x,y)$ et $min(x,y)$:



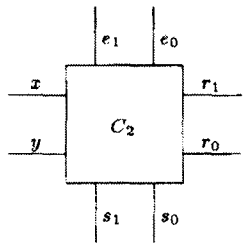
2 – Construire un circuit logique C_1 prenant en entrées deux bits d'information x et y et fournissant en sorties les valeurs r_1, r_0, s_1 et s_0 telles que :

- $(r_1,r_0) = (x,y)$ et $(s_1,s_0) = (1,0)$ si $x > y$;
- $(r_1,r_0) = (x,y)$ et $(s_1,s_0) = (1,1)$ si $x = y$;
- $(r_1,r_0) = (y,x)$ et $(s_1,s_0) = (0,1)$ si $x < y$.



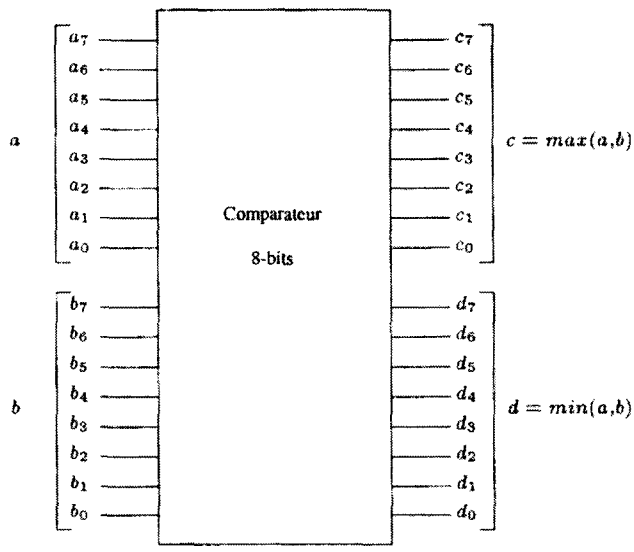
3 – Construire un circuit logique C_2 prenant en entrées quatre bits d'information e_1, e_0, x et y et fournissant en sorties les valeurs r_1, r_0, s_1 et s_0 telles que :

- $(r_1,r_0) = (x,y)$ et $(s_1,s_0) = (1,0)$ si $(e_1,e_0) = (1,0)$;
- $(r_1,r_0) = (x,y)$ et $(s_1,s_0) = (1,0)$ si $(e_1,e_0) = (1,1)$ et $x > y$;
- $(r_1,r_0) = (x,y)$ et $(s_1,s_0) = (1,1)$ si $(e_1,e_0) = (1,1)$ et $x = y$;
- $(r_1,r_0) = (y,x)$ et $(s_1,s_0) = (0,1)$ si $(e_1,e_0) = (0,1)$;
- $(r_1,r_0) = (y,x)$ et $(s_1,s_0) = (0,1)$ si $(e_1,e_0) = (1,1)$ et $x < y$;
- on ne s'intéresse pas au cas où $(e_1,e_0) = (0,0)$.



On appelle C_3 le circuit logique C_2 dans lequel on a supprimé les sorties s_1 et s_0.

On appelle *comparateur 8-bits* un circuit logique prenant en arguments deux entiers positifs ou nuls $a = \overline{a_7 \cdots a_0}^2$ et $b = \overline{b_7 \cdots b_0}^2$ strictement inférieurs à $2^8 = 256$ et fournissant en sorties les deux entiers positifs $c = \overline{c_7 \cdots c_0}^2$ et $d = \overline{d_7 \cdots d_0}^2$ tels que $c = max(a,b)$ et $d = min(a,b)$. Un tel circuit logique peut être représenté graphiquement ainsi :



4 – À l'aide des circuits logiques C_1, C_2 et C_3, construire un comparateur 8-bits.

FIN DU PROBLÈME DE LOGIQUE DES PROPOSITIONS

FIN DE L'ÉPREUVE