Semaine 5 : Série d'exercices sur la représentation de l'information [Solutions]

1 Nombre de bits

1. Les mois d'une année = 12 donc requiert au minimum 16 combinaisons, c.-à-d. 4 bits.
2. Les jours d'un mois = max 31 donc requiert au minimum 32 combinaisons, c.-à-d. 5 bits.
3. L'ensemble des symboles utilisés pour les nombres romain jusqu'à mille =I,V,X,L,C,D,M = 7 donc requiert au minimum 8 combinaisons, c.-à-d. 3 bits.
4. Pour les 10'000 étudiants à l'EPFL, utilisons quelques puissances de 2 bien connues pour calculer un ordre de grandeur. Par exemple 2^{10} vaut 1024, donc 16×2^{10} va suffire, c.-à-d. $2^{(4+10)} = 2^{14}$. 14 bits seront donc suffisants. Si nécessaire, un calcul exact nous donne $2^{13} = 8'192$ qui n'est en effet pas suffisant. Il faut donc bien 14 bits.
5. Chaque habitant de la planète : il faut encore une fois utiliser quelques puissances de 2 bien connues. Par exemple 2^{30} vaut un peu plus que 10^9, donc 8×2^{30} va suffire, c.-à-d. $2^{(3+30)} = 2^{33}$; c.-à-d. 33 bits.

Méthode

Comme on utilise des éléments binaires pour représenter les combinaisons distinctes, le nombre de combinaisons pour n bits est 2^n.

Pour cet exercice on fait l'inverse, on cherche le nombre de bits n pour pouvoir représenter au moins K combinaisons distinctes. Si K est une puissance entière de 2, on obtient directement cette puissance entière n en calculant le log de base 2 de K : $\log_2(K) = \log_2(2^n) = n \log_2(2) = n$.

Si par contre K n'est pas une puissance entière de 2, on peut malgré tout exprimer K comme une puissance m de 2, avec m compris entre deux entiers consécutifs. La valeur de m est obtenue comme précédemment en prenant le log de base 2 de K. Le nombre de bits recherché n est l'entier immédiatement supérieur à m.

2 Représentation des entiers naturels et décimaux positifs

1. Les bits à droite de la virgule représentent des puissances négatives de 2 en commençant par 2^{-1}, etc.
   10.1 donne $2 + 2^{-1} = 2.5$,
   1.01 donne $1 + 2^{-2} = 1.25$,
   0.101 donne $2^{-1} + 2^{-3} = 0.5 + 0.125 = 0.625$.

   Remarquez que ces nombres ont été divisés par la base (c.-à-d. par 2) à chaque fois, ce qui correspond à un décalage d'un cran vers la gauche de la virgule.

2. Soit X en décimal, on cherche son motif binaire inconnu et noté $0.b_1b_2b_3b_4$.
L'expression développée de X en binaire est donc $X = b_1 \times 2^{-1} + b_2 \times 2^{-2} + b_3 \times 2^{-3} + b_4 \times 2^{-4}$.

Les étapes sont :

1. Multiplier par 2 : $2 \times X = b_1 + b_2 \times 2^{-1} + b_3 \times 2^{-2} + b_4 \times 2^{-3}$.
2. Prendre la partie entière : c'est b_1, ce qui nous donne donc bien le premier bit.
2. Conserver maintenant uniquement la partie fractionnaire, c.-à-d. $b_2 \times 2^{-1} + b_3 \times 2^{-2} + b_4 \times 2^{-3}$. Si elle est nulle, la conversion est finie. Sinon, reprendre en (1) avec cette quantité.

Si X vaut 0.375 l'algorithme se déroule comme suit :

(a) $2 \times X$ vaut 0.750.
(b) b_1 vaut 0.
(c) La partie fractionnaire 0.750 n'est pas nulle : on continue.

(a) 2×0.750 vaut 1.5.
(b) b_2 vaut 1.
(c) La partie fractionnaire 0.5 n'est pas nulle : on continue.

(a) 2×0.5 vaut 1.
(b) b_3 vaut 1.
(c) La partie fractionnaire 0. est nulle, c'est fini ; donc b_4 vaut 0.

L'application de cette méthode à la quantité « un dixième » vous donne l'écriture vue en cours qui comporte un nombre infini de bits car un même motif se répète. En bref il n'y pas de représentation exacte de cette quantité en binaire (voir aussi le dernier exercice)

3 Domaine couvert des entiers positifs et négatifs

1. Conversions : motifs binaires sur 8 bits

(a) — `00000110` est positif, de valeur $4 + 2 = 6$.
— `11111001` est négatif car son bit de poids fort (celui le plus à gauche) est à 1. Pour connaitre sa valeur absolue, on calcule son opposé en prenant son complément à deux, c.-à-d. le complément à un, auquel on ajoute 1.
`11111001`
`00000110` Cp à 1
`00000001` +1
`00000111` ce qui vaut 7, donc le motif initial représente -7.
— `10000110` est négatif car son bit de poids fort est à `1`. Pour connaitre sa valeur absolue, on calcule son opposé en prenant son complément à deux, c.-à-d. le complément à 1 auquel on ajoute 1.
`10000110`
`01111001` Cp à 1
`00000001` +1
`01111010` ce qui vaut $2 + 8 + 16 + 32 + 64 = 122$, donc le motif initial représente -122.

(b) — 0 = `00000000`.
— -12 calculer d'abord 12 en binaire, puis prendre son complément à deux
`00001100` 12
`11110011` Cp à 1
`00000001` +1
`11110100` Cp à 2 = représentation de -12.
— -1 calculer d'abord 1 en binaire, puis prendre son complément à deux
`00000001`
`11111110` Cp à 1
`00000001` +1
`11111111` Cp à 2 = représentation de -1.
— 127_{10} = `01111111` = maximum des nombres positifs pour 8 bits.
— -128_{10} = `10000000` = minimum des nombres négatifs pour 8 bits. L'opposé usuel de ce nombre (128_{10}) n'a pas de représentation sur 8 bits et l'opposé du 8 bits de `10000000` (en complément à deux) est `10000000` : -128_{10} est son propre opposé sur 8 bits.

(c) Conclusion : le domaine n'est pas symétrique. Il faut surveiller le cas singulier où on demande l'opposé du min négatif, car cela donnerait un résultat incorrect.

2. (a) La valeur 64 est représentable avec ce motif binaire : `01000000`.

`01000000` 64
`01000000` +64
`10000000` donne -128.

(b) Il s'agit d'un dépassement de capacité par sortie du domaine couvert : l'addition de deux nombres positifs donne un nombre négatif, ce qui est incorrect.

(c) Le dépassement de capacité (par perte de la retenue) apparaît chaque fois qu'on ajoute :
— un nombre positif et un nombre négatif et que le résultat est positif ;
— deux nombres négatifs.

(d) Cela ne pose pas forcément problème : si le résultat est correct car c'est le bit de signe (8e bit) qui « absorbe » le dépassement de capacité sur les 7 bits de poids faible : la retenue est perdue au dela du bit de signe. C'est la *raison d'être* de l'utilisation du complément à deux.

Cette représentation ne pose problème que si l'on sort du domaine couvert :
— addition deux de nombres de même signe et résultat de signe opposé ;
— opposé du plus grand négatif.

4 Représentation des nombres flottants, précision absolue et relative

1. Avec 2 bits d'exposant, on aura pour la forme normalisée « $2^{\text{exposant}} \times 1$,mantisse », $2^2 = 4$ exposant différents possibles, donc 4 intervalles successifs dans chacun desquels les nombres représentés auront le même écart entre eux. Avec 3 bits de mantisse, il y a $2^3 = 8$ nombres par intervalle.

Si l'on ne considère pas l'optimisation du cas particulier de l'exposant 0 (transparents 41 à 43 du cours), alors le minimum est donné par : $2^0 \times 1.0$, c.-à-d. 1.

Si l'on considère ce cas particulier (option), alors le mimum est $2^{P+1} \times 0.000$, c.-à-d. 0.

Pour le maximum :
Exposant = 11 c.-à-d. 3,
Mantisse = 111 c.-à-d. 1.111_2, soit 0.875_{10}.

Le max est donné par : $2^3 \times 1.875 = 8 \times 1.875 = 15$ pour la version simple et $2^{3+P} \times 1.875 = 2^{3-1} \times 1.875 = 4 \times 1.875 = 7.5$ pour la version avec exposants décalés (en prennant $P = -(2^{2-1}-1) = -1$).

Les valeurs représentées par la forme $2^{exposant} \times 1.$mantisse sont :

1	1.125	1.25	1.375	1.5	1.625	1.75	1.875
2	2.25	2.5	2.75	3	3.25	3.5	3.75
4	4.5	5	5.5	6	6.5	7	7.5
8	9	10	11	12	13	14	15

Si l'on considère (option) le cas particulier de l'exposant 0 (transparent 41 à 43 du cours) avec $P = -(2^{2-1} - 1) = -1$, alors on a comme valeurs : 0 ($2^0 \times 0.000$). 0.125 ($2^0 \times 0.001$). 0.25 ($2^0 \times 0.010$). 0.375 ($2^0 \times 0.011$). 0.5 ($2^0 \times 0.100$). etc. jusque 0.875 ($2^0 \times 0.111$) pour la première ligne ci-dessus, puis toutes les autres valeurs divisées par 2.

2. L'erreur absolue n'est pas constante : c'est au pire la valeur de l'écart entre deux nombres représentés successifs si l'on arrondi les nombres vers le bas (troncature) (si l'on arrondi au plus proche, alors c'est la moitié de cet écart) ; cette valeur maximale double quand on passe d'un intervalle au suivant (par ordre croissant).
3. Avec la représentation de base (sans cas particulier de l'exposant 0), on aura dans l'ordre des erreurs absolues maximales de : 0.125 0.25 0.5 1
4. L'erreur relative maximum est obtenue en début d'intervalle, p.ex. lorsque l'on représente 1.124999 par 1, ou 2.24999 par 2, etc., ce qui correspond à environ [1] le poids faible de la mantisse (c.-à-d. son bit le plus à droite), soit $2^{-3} = 0.125$ c.-à-d. 12.5%. Cette erreur relative maximum est la même pour les 4 intervalles (sans cas particulier).

   Dans le cas où l'on ajoute le cas particulier de l'exposant 0 (option), l'erreur relative maximum n'est par contre pas la même pour l'exposant 0 que pour les autres exposants. On a alors $2^{-3}/(0+2^{-3}) = 100\%$ pour ce cas particulier.

Pour aller plus loin...

Montrons que l'écriture binaire de un dixième est `0.000110011...`, c.-à-d. $\texttt{0.}\overline{\texttt{0011}}$:

$$\begin{aligned}
\texttt{0.}\overline{\texttt{0011}}_2 &= \texttt{0.1}_2 \times \texttt{0.}\overline{\texttt{0011}}_2 = \frac{1}{2} \times \texttt{0.}\overline{\texttt{0011}}_2 \\
&= \frac{1}{2} \sum_{i=1}^{\infty} \frac{1}{2^{4i-1}} + \frac{1}{2^{4i}} = \frac{1}{2} \sum_{i=1}^{\infty} (2+1) \times \frac{1}{2^{4i}} \\
&= \frac{3}{2} \times (\frac{1}{1-\frac{1}{2^4}} - 1) = \frac{3}{2} \times \frac{1}{15} = \frac{1}{10}
\end{aligned}$$

1. En toute rigueur, ce n'est pas $2^{-3}/1$ mais $2^{-3}/(1+2^{-3}) = 1/9$, soit 11.1%.