Bit à bit

1 Premier exemple : adresse IP v4

Les adresses des ordinateurs en format IP v4 sont composées de quatre octets : c'est-à-dire qu'une adresse est la donnée de quatre nombres entiers que l'on peut écrire en base 2 à l'aide d'un octet (donc de 8 bits). On sépare généralement ces quatre nombres par des points. On peut aussi voir cette adresse comme un nombre en binaire sur 32 bits.

1. Hier soir, l'adresse de mon ordinateur (telle que donnée par le site `http://www.whatismyip.com/`) était 90.21.227.33

   Ecrire cette adresse en binaire : quatre nombres en binaire sur 8 bits (on complètera donc par des zéros à gauche si nécessaire). On note a le nombre en binaire sur 32 bits obtenu.

L'adresse est constituée de deux parties :
- l'adresse du sous-réseau
- l'adresse de l'hôte dans ce sous-réseau

Pour savoir associer une adresse de sous-réseau et une adresse d'hôte à une adresse IP v4, il nous manque une donnée : le masque de sous-réseau. Le masque de réseau indique quels sont les bits qui servent à lire l'adresse de sous-réseau (il faut conserver les bits de l'adresse correspondant aux 1 du masque et mettre des 0 aux bits de l'adresse correspondant aux 0 du masque), et quels bits servent à lire l'adresse de l'hôte (il faut conserver les bits de l'adresse correspondant aux 0 du masque et mettre des 0 aux bits de l'adresse correspondant aux 1 du masque).

2. On prend comme masque de sous-réseau 255.255.224.0 et on conserve l'adresse IP v4 de la question 1.
   (a) Ecrire ce masque en binaire. On note m le nombre en binaire sur 32 bits obtenu.
   (b) Déduire l'adresse de sous-réseau et l'adresse de l'hôte de ma machine hier soir en binaire, puis en décimal (on note respectivement s et h les nombres en binaire sur 32 bits obtenus)

2 Les opérateurs bit à bit

Les opérations que l'on vient de faire sont très simples à réaliser par un ordinateur, car il s'agit d'opérations binaires de base, effectuées sur chacun des bits de l'adresse et du masque.

Rappel : si b est un booléen, alors $b \wedge 1 = b$ et $b \wedge 0 = 0$. Ainsi, il suffit de faire un ET bit à bit entre l'adresse IP et le masque pour obtenir l'adresse de sous-réseau.

En Python, le ET bit à bit s'effectue à l'aide du `&`. Par exemple, `13 & 21 = 5` car $13_{10} = 1101_2$ et $21_{10} = 10101_2$. Ainsi :

$$\begin{array}{cccccc} & & 1 & 1 & 0 & 1 \\ \& & 1 & 0 & 1 & 0 & 1 \\ \hline = & 0 & 0 & 1 & 0 & 1 \end{array}$$

3. En Python, taper `90 & 255`, `21 & 255`, `227 & 224` et `33 & 0` et vérifier que l'obtient les mêmes résultats qu'à la question 2 pour s.

Afin de calculer facilement l'adresse de l'hôte, il faudrait faire un ET bit à bit... mais avec le masque « inversé », contenant des 0 là où le masque contenait des 1 et inversement. L'opérateur binaire qui permet de changer un 0 en 1 et inversement est le NON : $\neg$. Et l'opérateur NON bit à bit se note `~`.

Le NON du masque s'obtient donc en inversant chacun des bits, en Python avec le `~`. Étant donné que l'on travaille sur des nombres de 8 bits, il faut taper `~n + 256` pour obtenir le NON sur 8 bits du nombre n.

4. Déterminer l'inverse du masque à l'aide de Python et de l'opérateur `~`.
5. Déduire une manière simple de retrouver h à l'aide de Python.

Il existe ensuite le OU bit à bit, qui fonctionne comme on l'imagine : en faisant un $\vee$ booléen (le ou inclusif) bit à bit. C'est l'opérateur | sous Python. Par exemple :

$$\begin{array}{cccccc} & & 1 & 1 & 0 & 1 \\ | & 1 & 0 & 1 & 0 & 1 \\ \hline = & 1 & 1 & 1 & 0 & 1 \end{array}$$

6. Vérifier que $s \mid h = a$.

Enfin on peut présenter le XOR bit à bit, qui fonctionne en faisant un xor booléen (le ou exclusif). C'est l'opérateur ^ sous Python. Il est utile en cryptographie car réversible : si a et b sont deux nombres, (a^b)^a = b. Par exemple :

$$\begin{array}{cccccc} & & 1 & 1 & 0 & 1 \\ \hat{} & 1 & 0 & 1 & 0 & 1 \\ \hline = & 1 & 1 & 0 & 0 & 0 \end{array} \qquad \begin{array}{cccccc} & 1 & 1 & 0 & 0 & 0 \\ \hat{} & 1 & 0 & 1 & 0 & 1 \\ \hline = & 0 & 1 & 1 & 0 & 1 \end{array}$$

3 Utilisation pratique

En pratique, lorsque l'on désire utiliser toute la puissance des opérateurs bit à bit, on a aussi besoin d'évaluer certains bits particuliers. Dans cette section, nous allons nous intéresser à un système de validation de la première année de licence. Il s'agit, lors de la première année de licence, de valider 60 crédits (ECTS). En arrivant à la faculté, on a le choix entre 20 cours différents, qui permettent chacun de valider 6 crédits si on réussit l'examen final. Bien sûr les horaires de chacun de ces cours font qu'on ne peut pas les suivre tous les 20, mais on peut en suivre plus que 10 pour avoir un peu de marge : par exemple si on s'inscrit à 12 cours, on peut rater 2 examens sans que cela soit grave !

À la fin de l'année, les notes des étudiants aux différents examens sont traitées par un ordinateur : si la note est de 10 ou plus, l'examen est réussi, le cours validé (donnant donc droit à 6 crédits). Sinon, le cours n'est pas validé.

Chaque étudiant se voit donc associer par la machine un nombre (s'écrivant en binaire sur 20 bits). Le bit numéro i (on compte de droite à gauche, en commençant à 0 !) est associé au cours numéro i : si l'étudiant n'a pas suivi le cours, ou s'il a raté son examen, le bit est à 0. S'il a suivi le cours et réussi l'examen, le bit est à 1.

7. Michel se voit associer le nombre 192 693. Doit-il se réjouir ?

Chaque étudiant peut consulter sa réussite ou non à un cours. Il rentre le numéro du cours, et on lui répond 0 ou 1. Par exemple :

- si Michel rentre le numéro 2, on lui dit qu'il a réussi l'examen du cours numéro 2 ;
- s'il rentre le numéro 3, on lui dit qu'il ne s'est pas inscrit ou qu'il n'a pas réussi l'examen du cours numéro 3.

Pour ce faire, on utilise l'opérateur de décalage vers la gauche : <<. Si on tape $a << b$, cela va décaler tous les bits du nombre a de b bits vers la gauche (en mettant donc b bits nuls à droite). Par exemple $13 << 3 = 104$ car $13_{10} = 1101_2$, en le décalant de 3 vers la gauche cela donne $1101000_2 = 104_{10}$.

En pratique on utilise le plus souvent le décalage du simple nombre 1. Par exemple. $1 << 3 = 8$. Or $8_{10} = 1000_2$: il est donc maintenant très simple de tester si l'élève a réussi ou pas le 3$^{\text{e}}$ cours : il suffit de faire le ET bit à bit de son nombre avec $1 << 3$: si l'on tombe sur 0, c'est que le 3$^{\text{e}}$ bit était à 0, donc il n'a pas réussi ; si l'on tombe sur 1000_2, c'est que le 3$^{\text{e}}$ bit était à 1, donc il a réussi. Effectivement le ET bit à bit ne conserve que les bits communs aux deux nombres, et met 0 autre part !

8. Ecrire un programme Python qui prend en entrée le nombre d'un étudiant, et un numéro de cours, et répond si l'étudiant a réussi ou pas à ce cours.