Représentation des données : binaire, booléen, hexadécimal

1ère NSI - Travaux dirigés

1. Nombres en binaires

Conversion binaire vers décimal.

1. Donnez les valeurs entières décimales représentées par les nombres :
   - `0b101`
   - `0b10101`
   - `0b0101`
   - `0b00101`
2. Comment savoir qu'un entier représenté en binaire est divisible par 2 ? Par 4 ?

Conversion décimal vers binaire.

3. En utilisant les division successives par 2, donner la représentation binaire des nombres 14 et et 27.
4. En utilisant les soustractions de puissances de 2, donner la représentation binaire des nombres 13 et 30.
5. On considère des entiers représentés sur 1 octet. Quel est le plus grand entier représentable ? Réaliser l'addition binaire entre $a = 0b11011001$ et $b = 0b11101100$.

   Qu'advient-il du premier bit lors de cette addition ?
6. Quelle est la représentation binaire d'un nombre de la forme $2^k - 1$? De la forme 2^k ?
7. En remarquant que $2\,048 = 2^{11}$, donner la représentation binaire de 2021.

2. Capacité

Parmi les additions suivantes, lesquelles vont provoquer un dépassement de capacité lorsque les nombres sont encodés sur 8 bits ?

- 1111 1011 + 1001 1111
- 1001 1011 + 0111 1011
- 0011 1011 + 1001 1001
- 1010 1011 + 0001 0100

3. Masques de sous-réseau

Les adresses IPv4 sont codées sur **32 bits**. En notation décimale on les représente par 4 nombres séparés par des points. Par exemple : 192.168.100.2

L'adresse IPv4 est constituée de deux sous parties utilisant la même représentation : le sous-réseau et l'hôte. On utilise des **masques** constitués d'une suite de 1 suivi d'une suite de 0. Il y a 32 masques de sous réseau possibles. Par exemple : 255.255.255.0

Pour obtenir le sous réseau on applique l'opérateur ET entre les notations binaires de l'adresse IP et du masque de sous-réseau.

On réalise ce ET bit par bit.

Pour obtenir l'adresse de l'hôte, on applique l'opérateur ET entre les notations binaires de l'adresse IP et la négation (NON) du masque.

1. Vérifier que le code binaire correspondant à l'adresse 192.168.0.2 est : `11000000.10101000.00000000.00000010`
2. Calculer le code binaire du masque 255.255.255.0.
3. Calculer l'adresse binaire du sous réseau puis sa forme décimale.
4. Calculer l'adresse binaire de l'hôte puis donner sa forme décimale.

4. Électronique : premier pas vers le calcul

On emploie en électronique des *portes logiques* correspondant aux opérations booléennes et à des combinaisons de celles-ci.

Voici, par exemple, les portes ET et XOR :

- porte `ET` :

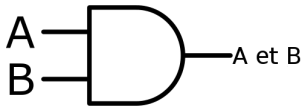

Figure 1: porte ET

- Porte `XOR` :

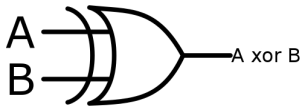

Figure 2: porte XOR

Rappel : tables de vérité des opérateurs `ET` et `XOR`:

A	B	A ET B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Les circuits électroniques calculent des fonctions logiques de l'algèbre de Boole.

Pour chacun des opérateurs logiques évoquées ci-dessus (et d'autres) il existe donc des portes logiques appelés *porte ET*, *porte NON*, etc. Les valeurs *vrai* et *faux* sont représentées par deux niveaux de tension, *haut* et *bas*.

Demi additionneur

Un circuit de type *porte ET* dispose donc de deux entrées et une sortie et la valeur du niveau de tension en sortie dépend des niveaux de tension appliquées à chaque entrée, en respectant la table de vérité du ET. Les portes peuvent être connectées entre elles pour réaliser des **circuits logiques** et on peut ainsi réaliser des calculs. Prenons l'exemple de ce circuit :

demi-additionneur

Il est appelé *demi-additionneur* car il réalise l'addition de 2 bits (**A** et **B**), le résultats de cette somme est représentée par **S** et la retenue éventuelle par **R**.

Étude du demi-additionneur

1. Vérifiez, avec une table de vérité, que **S** et **R** correspondent bien aux valeurs de la somme et de la retenue sur 1 bit de **A** et **B**.

A	B	A et B	A xor B	Retenue	Somme
0	0				
0	1				
1	0				
1	1				

5. Hexadécimal

Binaire ↔ Hexadécimal

1. On considère le nombre binaire suivant : `0b 1100 0101 1110 1010`.
    1. De combien d'octets est-il composé ?
    2. Convertir chaque paquet de 4 bits en un chiffre hexadécimal.
    3. Donner la représentation hexadécimale du nombre de départ.
2. Dans l'autre sens. On considère le nombre hexadécimal suivant : `0xA8`
    1. De combien d'octets est-il composé ?
    2. Convertir chaque octet en binaire et en déduire la représentation binaire du nombre de départ.

Décimal ↔ Hexadécimal

3. On part cette fois d'un entier décimal : `3452`. En réalisant les divisions successives par 16, donner la représentation hexadécimale de ce nombre.
4. *Code couleur.* En vous référant à la synthèse additive des couleurs, donner le code hexadécimal d'une couleur dont les niveaux respectifs de rouge, vert, bleu sont 80%, 50% et 20%.

6. Espace en mémoire

Intéressons nous à l'espace occupé par un nombre en mémoire ainsi qu'à l'espace occupé par le résultat d'une opération.

Les nombres sont enregistrés dans la mémoire d'un ordinateur *en base 2*, ils occupent *au moins* une taille égale à leur nombre de bits.

Par exemple, `32 = 0b100000` et `63 = 0b111111`. Il faut *au moins* 6 symboles pour stocker ces nombres. Et c'est le cas de tous les nombres entre 32 et 63 inclus.

On se pose la question suivante : quelles sont les **tailles** des résultats de la somme et du produit ?

On suppose connaître la taille en mémoire de deux entiers, peut-on en dire autant de leur somme et de leur produit ?

Somme

Partons d'un constat simple.

- Le plus grand entier occupant 6 bits est `63 = 0b111111`.
- Le suivant est `64 = 0b1000000` qui occupe 7 bits.
- La plus grande somme qu'on puisse réaliser de deux entiers de 6 bits est : `63 + 63 = 0b1111110`

1. Quel est le nombre maximal de bits de la somme de deux entiers occupant 6 bits ?

Produit

On se pose cette question : quelle est la taille (= nombre de bits) du résultat d'un produit ?

Raisonnons avec les puissances 2.

Par exemple $64 \times 32 = 2^6 \times 2^5 = 2^{11} \dots$

Qu'est ce que cela nous dit ?

Que la taille en mémoire occupée par le produit de d'entier sur 6 bits par un entier sur 5 bits est toujours inférieure ou égale à 11 bits.

Un exemple pour éclairer :

$23 \times 42 = 966$

`23 = 0b10111` : 5 bits. `42 = 0b101010` : 6 bits. Leur produit va occuper 11 bits ou moins de 11 bits.

En effet `966 = 0b11 1100 0110` : 10 bits.

2. On dispose d'un ordinateur qui peut stocker des entiers naturels occupant **au maximum 7 bits.**
   - Quel est le plus grand entier N qu'il puisse stocker en mémoire ?
   - Quelle taille devrait occuper le **carré** de cet entier ? On parle de *dépassement de capacité*.
   - Si A est un entier sur 5 bits, quelle taille maximale donner à B pour être certain que leur produit ne dépasse pas la taille limite ?

7. D'une base à l'autre

Cet exercice prépare la séance de travaux dirigés durant laquelle nous allons implémenter les changements de base.

1. Nous avons présenté deux algorithmes pour exprimer un nombre dans une base quelconque. Quel est celui qui semble le plus simple à implémenter ?
2. **Binaire.** On choisit d'implémenter l'algorithme des divisions successives.
   1. Proposer la signature d'une fonction Python permettant d'exprimer un entier dans une base quelconque.
   2. Convertir l'algorithme présenté en cours en Python. 3.**Base quelconque**.
   3. Quelle difficulté présente la conversion d'un nombre entier dans une base quelconque, comme 17 (par exemple) ?
   4. On décide d'utiliser les lettres (minuscules) pour compléter les chiffres usuels. Ansi, en base 13 on utiliserait `0123456789abc c` représente le *chiffre* douze. Dans cette base, $b2_{13} = 11 \times 13 + 2$
      1. Proposer un moyen d'associer facilement un nombre à son chiffre.
      2. Utiliser l'algorithme des divisions successives pour écrire une fonction Python qui réalise la conversion dans une base quelconque.