TP 10 - Représentation des nombres

Dans ce TP, on aborde la représentation des nombres dans différentes bases et leurs implémentations concrètes en informatique.

Dans tout ce TP, sauf mention contraire, il est interdit d'utiliser les fonctions `int`, `str` ou `list`. De manière générale, il est interdit de transformer le type de l'argument d'une fonction. Si la fonction prend en entrée un entier, elle travaille sur cet entier.

La première partie est à bien comprendre et à savoir manipuler correctement. La deuxième est au programme mais il s'agit plus de culture générale.

1 Systèmes de numération

1.1 Bases

On utilise naturellement la base 10, appelée **décimale**, et on écrit par exemple 2 013 signifiant "deux milliers, zéro centaines, une dizaine et trois unités" c'est-à-dire $2\,013 = 2 \times 10^3 + 0 \times 10^2 + 1 \times 10^1 + 3 \times 10^0$.

On peut généraliser cette écriture pour une base $k > 1$ quelconque : $\sum_{i \geq 0} a_i \, k^i$ avec $0 \leq a_i \leq k - 1$.

Remarque. $k > 1$ *car on ne peut pas décomposer un nombre en puissances de* 0 *à l'aide d'aucun symbole et on ne peut pas décomposer un nombre non nul en puissance de* 1 *à l'aide du seul symbole* 0.

Pour indiquer qu'un nombre est écrit dans une base différente de 10, on ajoute k (la base utilisée) en indice. Par exemple, 213_5 est un nombre en base 5.

Remarque. *La base* 10 *n'a rien de particulier par rapport aux autres, elle est plus facile à utiliser pour nous simplement parce que nous y sommes habitués.*

1.2 La base 2 (binaire)

La base deux, appelée **binaire**, est le système de numération naturel pour un ordinateur qui, fondamentalement, ne manipule que des nombres binaires. Pour indiquer qu'un nombre est écrit en binaire, on le souligne plutôt que de mettre 2 en indice.

Exemple 1.
- $\underline{101110}$ en base 10 vaut $1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 47$.
- $\underline{10101010}$ en base 10 vaut $2^7 + 2^5 + 2^3 + 2^1 = 170$.

Question 1. Ecrire la décomposition binaire des 10 premiers entiers.

Question 2. Combien de nombres entiers positifs différents peut-on coder sur n bits ? Quel est le plus grand entier codé sur n bits ? Quel est le plus petit entier codé sur n bits ?

On peut remarquer que sur **un octet (i.e.** 8 **bits)**, on peut coder les nombres binaires à 8 chiffres donc on peut coder $2^8 = 256$ **entiers différents** (de 0 à 255). Avec 32 bits, on peut en représenter $2^{32} = 4\,294\,967\,296$, environ $4,3$ milliards (lorsque les mémoires vives ont commencé à dépasser 4Go, il est devenu nécessaire de passer au 64 bits afin de pouvoir adresser tous les octets de la mémoire avec un seul mot).

Question 3. Combien d'espace mémoire minimum (en octets) est nécessaire pour représenter 16 803 256 ?

La notation en binaire offre une possibilité très intéressante : pour multiplier un nombre par deux, il suffit de décaler tous les bits de ce nombre d'un rang vers la gauche, et de rajouter un zéro à droite. Par exemple, multiplier 0101 (cinq) par deux donne 1010 (dix)... Les multiplications effectuées par les ordinateurs sont basées sur ce principe.

1.3 Conversions entre la base 10 et les autres bases

1.3.1 Revenir en base 10

Pour passer d'une base k quelconque en base 10, il suffit de reprendre la forme générale $\sum_i a_i \, k^i$ et de faire le calcul "normalement", en prenant soin si c'est nécessaire de bien exprimer k en base 10.

Exemple 2.
- $\underline{10101010}$ en base 10 vaut $2^7 + 2^5 + 2^3 + 2^1 = 170$.
- $213_5 = 2 \times 5^2 + 1 \times 5^1 + 3 \times 5^0 = 58$.
- $219_{12} = 2 \times 12^2 + 1 \times 12^1 + 9 \times 12^0 = 309$.
- $219_{16} = 2 \times 16^2 + 1 \times 16^1 + 9 \times 16^0 = 537$.

Question 4. Convertir en base décimale les nombres suivants : $\underline{110010101}$, 314_5 et 147_{13}.

Question 5. Ecrire une fonction `binaireVersDecimal` prenant en entrée un entier `n` en binaire, donné par une liste de 0 et de 1 dont le chiffre de poids le plus faible est à la fin, et renvoyant son écriture décimale sous la forme d'un entier Python.

Question 6. Ecrire une fonction `versDecimal` prenant en entrée un entier Python k et un entier `n` écrit en base `k`, donné par une liste d'entiers compris entre 0 et $k-1$ dont le chiffre de poids le plus faible est à la fin, et renvoyant son écriture décimale sous la forme d'un entier Python.

1.3.2 Quitter la base 10 : algorithme de Horner

L'inverse est légèrement plus délicat. Pour passer de la base 10 à une base k quelconque, on utilise l'algorithme de Horner suivant : on effectue la division euclidienne du nombre à convertir par k, on note le reste (ce sera le dernier chiffre), on recommence avec le quotient, etc... jusqu'à arriver à un quotient nul : la décomposition est alors terminée.

Remarque. *Remarquons que cet algorithme devient plus clair si au lieu d'écrire* $a_0k^0+a_1k^1+a_2k^2+a_3k^3+...$, *on écrit* $a_0+k(a_1+k(a_2+k(a_3+k(...))))$.

Exemple 3. • On décompose 100 en base 2. Comme $100 = 50\times 2+0$, la décomposition binaire de 100 termine par un 0. Ensuite, comme $50 = 25\times 2+0$, la décomposition binaire de 100 termine par 00. Puis, comme $25 = 12\times 2+1$, la décomposition binaire de 100 termine par 100. Ensuite, comme $12 = 6\times 2+0$, la décomposition binaire de 100 termine par 0100. Puis, comme $6 = 3\times 2+0$, la décomposition binaire de 100 termine par 00100. Ensuite, comme $3 = 1\times 2+1$, la décomposition binaire de 100 termine par 100100. Pour finir, comme $1 = 0\times 2+1$, la décomposition binaire de 100 vaut 1100100.

• On décompose 58 en base 5. Comme $58 = 11\times 5+3$, la décomposition de 58 en base 5 termine par un 3. Ensuite, comme $11 = 2\times 5+1$, la décomposition se termine par 13. Pour finir, $2 = 0\times 5+2$, donc la décomposition de 58 en base 5 est 213.

• On décompose 309 en base 12. Comme $309 = 25\times 12+9$, la décomposition se termine par un 9. Ensuite, comme $25 = 2\times 12+1$, la décomposition se termine en 19. Puis, $2 = 0\times 12+2$ donc la décomposition de 309 en base 12 est 219.

• On décompose 537 en base 16. Comme $537 = 33\times 16+9$, la décomposition se termine par un 9. Ensuite, comme $33 = 2\times 16+1$, la décomposition se termine en 19. Puis, $2 = 0\times 16+2$ donc la décomposition de 537 en base 16 est 219.

Question 7. Ecrire 267 en binaire, en base 3, en base 5.

Question 8. Ecrire une fonction `decimalVersBinaire` prenant en entrée un entier `n` Python en décimal et renvoyant son écriture binaire sous la forme d'une liste de 0 et de 1.

Question 9. Ecrire une fonction `versBasek` prenant en entrées un entier `n` Python écrit en décimal et un entier `k` ; et renvoyant son écriture en base `k` sous la forme d'une liste.

2 Représentation des flottants

Pour les nombres non entiers, le codage utilisé s'appelle **"virgule flottante"**. L'idée est d'utiliser une représentation analogue à la notation scientifique, mais en base deux et non en base dix. La forme générale est $s\,m\,2^{n-d}$ où s est le signe codé sur un seul bit, m est la **mantisse** comprise entre 1 (inclus) et 2 (exclu), $n-d$ est l'exposant (entier) et $d = 2^{N-1}-1$ le décalage où N est le nombre de bits dans l'écriture de n.

2.1 Simple précision

Pour un flottant codé sur 32 bits (qualifié de **simple précision**, par rapport aux flottants codés sur 64 bits pour lesquels on parle de **double précision**), il y a 1 bit pour le signe, 8 bits pour l'exposant, 23 bits pour la mantisse et le décalage vaut $2^7-1 = 127$.

Exemple 4. Codons 21,625 sur 32 bits. Comme $21,625 = 1\times 2^4+0\times 2^3+1\times 2^2+0\times 2^1+1\times 2^0+1\times 2^{-1}+0\times 2^{-2}+1\times 2^{-3}$, on a $21,625 = \underline{10101.101} = \underline{1.0101101}*2^4$.

Ainsi le signe est positif, le bit correspondant est 0 ; la mantisse est constituée de 23 bits obtenus en complétant 0101101 avec des zéros et l'exposant $n-d = n-127 = 4$ donc $n = 131 = \underline{10000011}$. D'où 21,625 est codé par 0 10000011 01011010000000000000000.

Question 10. Coder 34,875 sur 32 bits.

Exemple 5. Considérons le flottant, codé sur 32 bits, 1 00100111 00100000000000000000000.

Ainsi le bit de signe est 1 donc le nombre est négatif ; son exposant $n = \underline{00100111} = 39$ donc $n - d = 39 - 127 = -88$ et sa mantisse est représentée par $\underline{001}$ i.e. $\underline{1.001} = 1 + 0 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} = 1,125$. On reconnaît le nombre $-1,125 \times 2^{-88}$.

Question 11. Ecrire en décimal le flottant codé sur 32 bits par 1 01000101 00101010000000000000000.

2.2 Problèmes d'arrondis

Tout d'abord, la précision et la capacité de représentation en virgule flottante sont limitées, même si (en double précision en particulier) elles s'avèrent suffisantes pour la plupart des applications. Cette précision est de l'ordre de $2^{-23} \simeq 10^{-7}$ en simple précision et de $2^{-52} \simeq 10^{-16}$ en double précisions (on a donc respectivement de l'ordre de 7 et 16 chiffres significatifs).

Mais le fait que le codage soit fondamentalement en base 2 et non en base 10 fait que des nombres décimaux comme 0,1 ne sont pas représentables de manière exacte en virgule flottante, et des problèmes d'arrondis vont se manifester dans des calculs apparemment simples. Il faut donc être attentif, du fait des arrondis il n'est pas équivalent de faire les deux calculs suivants : `(2**(-54)+1)-1` vaut `0.0` et `2**(-54)+(1-1)` vaut `5.551115123125783e-17` **Autrement dit, avec les flottants, l'addition n'est pas associative !**

Tout cela implique qu'**il ne faut jamais faire de comparaison d'égalité entre les flottants**, en particulier il est dangereux de tester si un réel est nul, il vaut mieux dans ce cas tester l'inégalité avec une majoration raisonnable, en dessous de laquelle on peut considérer que la valeur est nulle.

Question 12. On considère la suite u définie par $u_0 = 0,1$ et pour tout $n \geq 1$, $u_{n+1} = 11 \times u_n - 1$.

1. Que dire de la suite u ?
2. Ecrire une fonction `suite01(n)` calculant le $n^{ème}$ terme de cette suite u.
3. Utiliser la fonction précédente pour évaluer certains termes de la suite. Que remarquez-vous ?

Question 13. On considère la suite v définie par $v_0 = 0,125$ et pour tout $n \geq 1$, $v_{n+1} = 11 \times v_n - 1,25$.

1. Que dire de la suite v ?
2. Ecrire une fonction `suite0125(n)` calculant le $n^{ème}$ terme de cette suite v.
3. Utiliser la fonction précédente pour évaluer certains termes de la suite. Que remarquez-vous ?

Le problème d'arrondis vient du fait que $0,1$ n'est pas représentable exactement mais que $0,125$ l'est.