Devoir Maison n°1

À rendre le 5 novembre 2015

Exercice 1 (Exponentiation rapide)

1.(a) Écrire une fonction de la forme

```
def power(x,n): #...
```

qui prend deux arguments x et n de type `integer` et retourne x^n.
(*Remarque : bien entendu, on ne fera pas appel aux opérations* `**` *et* `^` *du langage Python.*)

1.(b) L'algorithme d'*exponentiation rapide* repose sur l'observation suivante : pour tout $x, n \in \mathbb{N}$,

$$x^n = \begin{cases} x^k \times x^k & \text{si } n = 2k \\ x^k \times x^k \times x & \text{si } n = 2k+1 \end{cases}$$

Écrire une fonction `quick_power` utilisant l'observation précédente.
(*Indice : on n'hésitera pas à programmer de façon récursive*)

1.(c) Comparer les complexités des deux algorithmes codés aux deux premières questions. On détaillera le calcul.

Exercice 2 (Grands entiers)

Habituellement, les variables de type `integer` sont stockés en mémoire sur 32 bits. C'est-à-dire que l'on ne peut utiliser ce type que pour les entiers entre 0 et $2^{32}-1$. Pour faire de l'arithmétique sur des entiers plus grands, on opte pour la technique suivante : un `BigInt` (pour *big integers*) sera simplement une liste de chiffres représentant la décomposition décimale de l'entier que l'on cherche à décrire. Par convention, le chiffre des unités sera le dernier de la liste (pour garder l'ordre de lecture habituel). Par exemple, dans notre formalisme,

— l'entier 3141592653589793 est représenté par

```
[3,1,4,1,5,9,2,6,5,3,5,8,9,7,9,3]
```

— l'entier 42 est représenté par `[4,2]`

— l'entier 0 est représenté par `[0]`

— etc.

On se propose, dans cet exercice, de coder quelques opérations basiques sur les `BigInt`.

2.(a) Écrire une fonction `BIstr` de type `BigInt` → `string` renvoyant dans une chaîne de caractère l'écriture décimale habituelle d'un grand entier. Par exemple :

```
BIstr([3,1,4,1,5,9,2,6,5,3,5,8,9,7,9,3])
```

doit être la chaîne de caractère `'3141592653589793'`

2.(b) Écrire une fonction `BIfrom_integer` de type `integer` → `BigInt` transformant un entier en grand entier. Par exemple, `BIfrom_integer(42)` doit renvoyer `[4,2]`.

2.(c) Écrire une fonction de la forme

```
def BIsum(m,n): #...
```

où m et n sont de type `BigInt` et qui renvoie le `BigInt` représentant la somme de m et de n.

2.(d) Écrire une fonction de la forme

```
def BIprod(m,n): #...
```

où m et n sont de type `BigInt` et qui renvoie le `BigInt` représentant le produit de m et de n.

2.(e) Écrire une fonction de la forme

```
def BIpower(x,n): #...
```

où x est de type `BigInt` et n de type `integer`, et qui renvoie le `BigInt` représentant x^n. On utilisera pour cela la méthode de l'exponentiation rapide. (Attention, n **n'est pas** un `BigInt` !)

Remarque : si vous essayez de jouer avec de très grands entiers dans la boucle intéractive de Python, vous verrez que Python arrive tout à fait à les gérer. En revanche, si vous checker le type des objets que vous manipulez (avec la fonction `type`), vous verrez qu'on passe à un moment (dépendant de la machine sur laquelle on se trouve) du type `int` au type `long`. Le type `long` est une implémentation des grands entiers au même titre que la nôtre (mais en bien plus efficace) !