I - TP 1

Révisions

1 Consignes

Pour récupérer les fichiers du tp, connectez-vous au site de la vie scolaire. Une archive vous a été envoyée, contenant les documents utiles pour le tp. Le fichier `tp_template.py` contient les codes du tp prétapés à compléter. Le fichier `tp_test.py` contient les différentes fonctions de test que les fonctons que vous écrivez doivent passer avec succès.

Vous sauvegarderez ces fichiers dans votre répertoire personnel, avec un chemin de la forme :
`NSI/Chapitre x - nom du chapitre/TP/TP1 - nom du tp/`
et vous renommerez le fichier `tp_templates.py` en `tp_nom_prénom.py`.
Ainsi pour ce premier TP votre répertoire de travail ressemblera à :

Résultat

```
NSI
   Chapitre 1 - Récursivité
       TP
           TP1 - Révisions
               1 - Révisions.zip
               tp_sujet.pdf
               tp_template.py
               tp_tests.py
```

2 Écriture binaire

Dans la mémoire de l'ordinateur, toutes les données (nombres, chaînes de caractères, mais aussi images, son...) sont stockées en utilisant uniquement des 0 et des 1. Pour représenter un nombre dans la mémoire de l'ordinateur, on utilise donc son écriture dans la base 2.

On rappelle que si un nombre n s'écrit $\overline{10110001}^2$ en base 2, alors cela veut dire que $n = 177$. On peut utiliser le tableau ci-dessous pour présenter le calcul.

Écriture en base 2	1	0	1	1	0	0	0	1	
Puissances de 2	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	Total
Termes à ajouter	128	0	32	16	0	0	0	1	177

On remarquera que dans cette convention d'écriture, le bit de poids fort (associé à 2^7) est le bit le plus à gauche dans l'écriture du nombre et que le bit de poids faible (aussi appelé bit de parité) est le bit le plus à droite dans l'écriture du nombre. On utilisera cette convention dans tous les exercices.

1 Déterminer l'écriture en base 2 des entiers compris entre 0 (inclus) et 17 (exclu).

Résultat

```
0 s'écrit en base 2 : [0]
1 s'écrit en base 2 : [1]
2 s'écrit en base 2 : [1, 0]
3 s'écrit en base 2 : [1, 1]
4 s'écrit en base 2 : [1, 0, 0]
5 s'écrit en base 2 : [1, 0, 1]
6 s'écrit en base 2 : [1, 1, 0]
7 s'écrit en base 2 : [1, 1, 1]
8 s'écrit en base 2 : [1, 0, 0, 0]
9 s'écrit en base 2 : [1, 0, 0, 1]
10 s'écrit en base 2 : [1, 0, 1, 0]
11 s'écrit en base 2 : [1, 0, 1, 1]
12 s'écrit en base 2 : [1, 1, 0, 0]
13 s'écrit en base 2 : [1, 1, 0, 1]
14 s'écrit en base 2 : [1, 1, 1, 0]
15 s'écrit en base 2 : [1, 1, 1, 1]
16 s'écrit en base 2 : [1, 0, 0, 0, 0]
```

2.1 Parité d'un nombre écrit en base 2

Écrire une fonction `est_pair` qui étant donné un tableau `bits` non vide dont les éléments appartiennent à $\{0;1\}$ détermine si le nombre dont l'écriture en base 2 est donnée par le tableau `bits` est un nombre pair. La fonction `est_pair` renverra `True` si c'est le cas, et `False` sinon.

Code python

```python
def est_pair(bits):
    """ [int] -> bool
    len(bits) > 0
    Détermine si le nombre dont l'écriture en base 2 est donnée par le
    ↪ tableau bits est pair.  """
    return bits[-1] == 0
```

Code python

```python
print(est_pair([0]))
print(est_pair([1]))
print(est_pair([1, 1, 0, 0, 1]))
```

Résultat

```
True
False
False
```

2.2 Nombre de bits d'un entier

2.2.1 Plus grand entier écrit avec k bits

Écrire une fonction `encadre` qui étant donné un nombre entier `k` détermine la valeur du plus grand entier que l'on puisse écrire en base 2 avec `k` bits.

Indication. Le plus grand entier que l'on puisse écrire en base 2 sur 4 bits est $\overline{1111}^2$, soit $1+2+4+8=15$. Plusieurs versions sont possibles :

- `encadre1` somme les k premières puissances de 2 à l'aide de l'algorithme de sommation ;

- `encadre2` effectue le même traitement en calculant la liste des 2^i pour $i \in \{0, \ldots, k-1\}$ à l'aide d'une liste en compréhension, puis utilise la fonction python `sum`.
- `encadre` utilise la formule mathématique vue en seconde : pour tout $n \in \mathbb{N}$,

$$1 + q + q^2 + \ldots + q^n = \frac{1 - q^{n+1}}{1 - q}$$

Code python

```
def encadre1(k):
    """ int -> int
    k > 0
    Détermine le plus grand entier que l'on puisse écrire avec k bits. """
    # On cherche à calculer : 1 + 2^1 + 2^2 + ...  + 2^{k - 1}
    somme = 0
    for i in range(k):
        somme += 2**i
    return somme

def encadre2(k):
    """ int -> int
    k > 0
    Détermine le plus grand entier que l'on puisse écrire avec k bits. """
    # On génère le tableau [1, 2, 4, 8, ..., 2**(k - 1)]
    termes = [2**i for i in range(k)]
    return sum(termes)

def encadre(k):
    """ int -> int
    k > 0
    Détermine le plus grand entier que l'on puisse écrire avec k bits. """
    # D'après le cours de maths :
    # 1 + 2^1 + 2^2 + ...  + 2^{k - 1} = 2^k - 1
    return 2**k - 1
```

Code python

```
print(encadre(4))
```

Résultat

```
15
```

2.2.2 Nombre de bits d'un entier

À l'aide de la fonction `encadre`, écrire une fonction `nombre_bits` qui étant donné un entier `n` détermine le nombre `k` de bits présents dans l'écriture en base 2 du nombre `n`. On ne se souciera pas de l'efficacité de la fonction `nombre_bits`.

Indication. Le nombre de bits `k` recherché est le plus petit nombre tel que `n <= encadre(k)`.

Code python

```
def nombre_bits(n):
    """ int -> int
    Détermine le nombre de bits de n dans son écriture en base 2 """
    nombre_bits = 1
    while encadre(nombre_bits) < n:
        nombre_bits += 1
    return nombre_bits
```

Code python

```python
print(nombre_bits(1))
print(nombre_bits(15))
print(nombre_bits(16))
```

Résultat

```
1
4
5
```

2.3 Entier maximal à nombre de bits fixés

Écrire une fonction `est_plus_grand_kbits` qui prend en entrée un tableau `bits` non vide de taille `k` dont les éléments appartiennent à $\{0;1\}$ et qui détermine si le tableau correspond au plus grand nombre entier que l'on puisse écrire en base 2 sur `k` bits.

Code python

```python
def est_plus_grand_kbits(bits):
    """ [int] -> bool
    k = len(bits) > 0
    Détermine si le tableau bits correspond au plus  grand entier que l'on
    puisse écrire sur k bits.  """
    # bits correspond au plus grand entier si il n'est constitué que de 1
    :
    # On parcourt tout le tableau :
    # - si jamais un des éléments est 0 on s'arrête et on renvoie False.
    # - si on a parcouru tout le tableau sans s'arrêter on renvoie True.
    for b in bits:
        if b == 0:
            return False
    return True
```

Code python

```python
print(est_plus_grand_kbits([1, 0, 1]))
print(est_plus_grand_kbits([1, 1, 0]))
print(est_plus_grand_kbits([1, 1, 1]))
```

Résultat

```
False
False
True
```

2.4 Base 2 vers base 10

Écrire une fonction `base2_vers_base10` qui étant donné un tableau `bits` non vide dont les éléments appartiennent à $\{0;1\}$ renvoie le nombre dont l'écriture en base 2 est donnée par le tableau `bits`.

Code python

```python
def base2_vers_base10(bits):
    """ [int] -> int
    Détermine le nombre entier dont l'écriture en base 2 est donnée par le
    tableau bits """
```

```python
    n = 0
    for i in range(len(bits)):
        n += bits[i]*2**(len(bits) - i - 1)
    return n
```

Code python

```python
print(base2_vers_base10([0]))
print(base2_vers_base10([1]))
print(base2_vers_base10([1, 0]))
print(base2_vers_base10([1, 0, 1, 0, 1, 0]))
```

Résultat

```
0
1
2
42
```

3 Algorithmes classiques

3.1 Recherche d'occurrences

3.1.1 Appartient

Écrire une fonction `appartient` qui étant donné un tableau (éventuellement vide) `tab` d'entiers et entier `e`, détermine si l'élément `e` est présent dans le tableau `tab`.

Code python

```python
def appartient(tab, e):
    """ [int], int -> bool
    Détermine si l'élément e est présent dans le tableau tab.  """
    for elem in tab:
        if elem == e:
            return True
    return False
```

Code python

```python
print(appartient([0, 1, 2, 3], 1))
print(appartient([0, 1, 2, 3], 4))
```

Résultat

```
True
False
```

3.1.2 Nombre d'occurrences

Écrire une fonction `nombre_occurrences` qui étant donné un tableau (éventuellement vide) `tab` d'entiers et un entier `e`, détermine le nombre de fois où l'élément `e` est apparait dans le tableau `tab` (on appelle ce nombre le nombre d'*occurrences* de l'élément `e`).

Code python

```python
def nombre_occurrences(tab, e):
    """ [int], int -> int
    Détermine le nombre d'éléments de tab qui sont égaux à e.  """
```

```python
    compte = 0
    for elem in tab:
        if elem == e:
            compte += 1
    return compte
```

Code python

```python
print(nombre_occurrences([0, 1, 2, 3, 1, 2, 1], 0))
print(nombre_occurrences([0, 1, 2, 3, 1, 2, 1], 2))
print(nombre_occurrences([0, 1, 2, 3, 1, 2, 1], 1))
```

Résultat

```
1
2
3
```

3.1.3 Indice des occurrences

Écrire une fonction `indices_occurrences` qui étant donné un tableau (éventuellement vide) `tab` d'entiers de type quelconque et un entier `e`, détermine la liste des indices (rangés par ordre croissant) des éléments de `tab` qui sont égaux à `e`.

Rappel. On pourra soit faire appel à la fonction `nombre_occurrences` pour initialiser le tableau d'indice des éléments égaux à `e` à la bonne taille, ou bien initialiser un tableau vide et ajouter les indices des éléments égaux à `e` au fur et à mesure à l'aide de l'instruction `t.append(elem)`.

Code python

```python
def indices_occurrences(tab, e):
    """ [int], int -> [int]
    Détermine les indices des occurrences de e dans le tableau tab. """
    indices = [0]*nombre_occurrences(tab, e)
    nb_occurrences = 0
    for i in range(len(tab)):
        if tab[i] == e:
            indices[nb_occurrences] = i
            nb_occurrences += 1
    return indices

def indices_occurrences(tab, e):
    """ [int], int -> [int]
    Détermine les indices des occurrences de e dans le tableau tab. """
    # Avec la méthode append
    indices = []
    for i in range(len(tab)):
        if tab[i] == e:
            indices.append(i)
    return indices
```

Code python

```python
print(indices_occurrences([0, 1, 2, 3, 1, 2, 1], 0))
print(indices_occurrences([0, 1, 2, 3, 1, 2, 1], 2))
print(indices_occurrences([0, 1, 2, 3, 1, 2, 1], 1))
```

Résultat

```
[0]
[2, 5]
[1, 4, 6]
```

3.2 Recherche de maximum

Écrire une fonction `maximum` qui étant donné un tableau non vide `tab` d'entiers détermine la valeur du plus grand des éléments de ce tableau, ainsi que le premier indice pour lequel ce maximum est atteint.

Code python

```python
def maximum(tab):
    """ [int], int -> int, int
    len(tab) > 0
    Détermine le maximum des éléments de tab, ainsi que le premier indice
    ↪ pour lequel ce maximum est atteint.  """
    record = tab[0]
    indice = 0
    for i in range(len(tab)):
        if tab[i] > record:
            record = tab[i]
            indice = i
    return record, indice
```

Code python

```python
print(maximum([1]))
print(maximum([1, 2, -1]))
print(maximum([1, 2, -1, 2]))
```

Résultat

```
(1, 0)
(2, 1)
(2, 1)
```