I - TP 1

Révisions

1 Consignes

Pour récupérer les fichiers du tp, connectez-vous au site du cours, http://florian.picard.forge.aeif.fr/terminale-nsi/. Vous trouverez dans la section correspondante (en base de page) tous les documents utiles pour le tp. 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/`

Vous créerez un fichier intitulé `tp.py` que vous remplirez avec le code des fonctions demandées dans l'énoncé. Ainsi pour ce premier TP votre répertoire de travail ressemblera à :

Résultat

```
NSI
   Chapitre 1 - Programmation orientée objet
       TP
           TP1 - Révisions
               tp_sujet.pdf
               tp.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).

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.  """
    pass
```

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 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.  """
    pass
```

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
```

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.  """
    pass
```

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.  """
    pass
```

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.  """
    pass
```

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.  """
    pass
```

Code python

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

Résultat

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

3.3 Tri par sélection

Écrire une fonction `tri_selection` qui prend en paramètre un tableau non vide `tab` de nombres entiers et qui trie ce tableau en place (c'est-à-dire que le tableau est modifié) par ordre croissant des valeurs. On utilisera l'algorithme suivant :

- On parcourt le tableau de gauche à droite :
  - on recherche le minimum du tableau entre cette position courante et la fin du tableau ;
  - on échange alors les 2 valeurs

Code python

```python
def tri_selection(tab):
    """ [int] -> [int]
    len(tab) > 0
    Trie en place le tableau tab, à l'aide de l'algorithme du tri par sélection """
    pass
```

Code python

```python
print(tri_selection([9,4,6,1]))
```

Résultat

```
[1, 4, 6, 9]
```