I - TP 1

Révisions

1 Consignes

Pour récupérer les fichiers du tp, connectez-vous au site du cours :

https://www.pisurquatre.xyz/nsi/

Vous trouverez sur le dépôt github du cours (https://github.com/marzikill/nsi-codes) tous les fichiers utiles pour la réalisation des différents tps : la version pdf du sujet (le fichier `tp.pdf`), fichiers de templates à compléter avec votre éditeur de texte favori (fichier `tp.py`), et éventuellement des fichiers annexes lorsque nécessaire.

Votre répertoire personnel de travail aura la même arborescence de fichier que le dépôt github.

Ainsi, vous sauvegarderez vos fichiers avec un chemin de la forme :
`NSI/Numéro du chapitre - nom du chapitre/Numéro du tp - nom du tp/fichier.extension`
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
 1 - Programmation orientée objet
   1 - Révisions
     tp.pdf
     tp.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.

Question 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

```
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

```
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

```
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

```
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
```

2.3 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

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

Code python

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

Résultat

```
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

```
def appartient(tab, e):
    """ [int], int -> bool
    Détermine si l'élément e est présent dans le tableau tab.  """
    pass
```

Code python

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

Résultat

```
True
False
```

Question 2. **1.** Modifier la fonction `appartient` en une fonction `appartient2` pour que celle-ci renvoie l'indice de la première occurrence de `e` dans `tab` si `e` est présent dans `tab`, `-1` sinon.

2. Modifier le code de la fonction `appartient2` en une fonction `appartient3` pour que celle-ci renvoie l'indice de la **dernière** occurrence de `e` dans `tab` si `e` est présent dans `tab`, `-1` sinon ?

3.1.2 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`.

Code python

```
def indices_occurrences(tab, e):
    """ [int], int -> [int]
    Détermine les indices des occurrences de e dans le tableau tab.  """
    pass
```

Code 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]
```

Question 3. En déduire une fonction `nombre_occurrences` qui étant donné un tableau `tab` et un élément `e` renvoie le nombre d'occurrences de l'élément `e` dans le tableau `tab`. Écrire correctement la docstring de cette fonction ainsi que sa documentation.

3.1.3 Occurrences

Écrire une fonction `occurrences` qui étant donné un tableau `tab` renvoie **un dictionnaire** structuré de la manière suivante :

- l'ensemble des clés du dictionnaire est l'ensemble des valeurs de `tab` ;
- à chaque clé `k` est associée la liste des indices des occurrences de `k` dans `tab`

Code python

```python
def occurrences(tab):
    """ [int] -> { int:[int] }
    Associe à chaque valeur de tab la liste des indices des occurrences de
    ↪ cette valeur """
    pass
```

Code python

```python
print(occurrences(['a', 'b', 'a', 'c']))
```

Résultat

```
{'a': [0, 2], 'b': [1], 'c':
↪ [3]}
```

Question 4. Écrire la fonction `occurrences` en ne réalisant **qu'un seul** parcours de `tab`.

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)
```

Question 5. Modifier la fonction `maximum` en une fonction `maximum2` afin que celle-ci renvoie la valeur du plus grand des éléments du tableau `tab` ainsi que le **dernier** indice pour lequel ce maximum est atteint.

3.3 Vérification de tri

Écrire une fonction `est_trie_croissant` qui étant donné un tableau `tab`, renvoie `True` si le tableau est trié par ordre croissant, `False` sinon.

Code python

```python
def est_trie_croissant(tab):
    """ [int] -> bool
    Détermine si le tab est trié par ordre croissant """
    pass
```

Code python

```python
print(est_trie_croissant([]))
print(est_trie_croissant([2, 4, 6]))
print(est_trie_croissant([3, 9, 6]))
```

Résultat

```
True
True
False
```