Algorithmique
Partiel n° 1 (P1)

Info-sup (s1)
Epita

3 Jan. 2017 - 10 : 00

Consignes (à lire) :

- ☐ Vous devez répondre sur **les feuilles de réponses prévues à cet effet**.
  - — Aucune autre feuille ne sera ramassée (gardez vos brouillons pour vous).
  - — Répondez dans les espaces prévus, **les réponses en dehors ne seront pas corrigées** : utilisez des brouillons !
  - — Ne séparez pas les feuilles à moins de pouvoir les ré-agrafer pour les rendre.
  - — Aucune réponse au crayon de papier ne sera corrigée.
- ☐ La présentation est notée en moins, c'est à dire que vous êtes noté sur 20 et que les points de présentation (2 au maximum) sont retirés de cette note.
- ☐ **Le code :**
  - — Tout code doit être écrit dans le langage `Python` (pas de C, Caml, Algo ou autre).
  - — **Tout code `Python` non indenté ne sera pas corrigé.**
  - — Tout ce dont vous avez besoin (fonctions, méthodes) est indiqué en **annexe** !
- ☐ Durée : 2h00

Garage et recherche

Exercice 1 (Piles et garage – *3 points*)

Imaginons un garage possédant deux voies d'entrée et une seule de sortie (voir figure 1). Pour garer sa voiture il existe 2 voies d'accès (*entrée e1* et *entrée e2*) et pour la ressortir une seule voie (*sortie s*).

Les mouvements possibles des voitures sont :

— une voiture peut entrer par l'*entrée e1*
— une voiture peut entrer par l'*entrée e2*
— une voiture ne peut sortir que par la *sortie s*
— une voiture ne peut ressortir que si elle est la dernière entrée (peu importe l'entrée)
— une voiture ne peut pas sauter par dessus une autre

Remarque : Il n'y a pas de problème de voitures se retrouvant face à face.

Le garage fonctionne comme une pile à double entrée.

Symbolisons une entrée à l'aide du couple $(v_{\textit{numéro du véhicule}}, e_{\textit{numéro de l'entrée empruntée}})$ et la sortie simplement par la lettre s.

On réalise la suite d'actions suivante :
$(v_1, e_1),(v_2, e_1),(v_3, e_2),(v_4, e_1),s,s,(v_5, e_2),(v_6, e_2),s,s,s,(v_7, e_1),(v_8, e_2),(v_9, e_2),s,s,s,s$

L'ordre de sortie des voitures est donc :
$v_4,v_3,v_6,v_5,v_2,v_9,v_8,v_7,v_1$

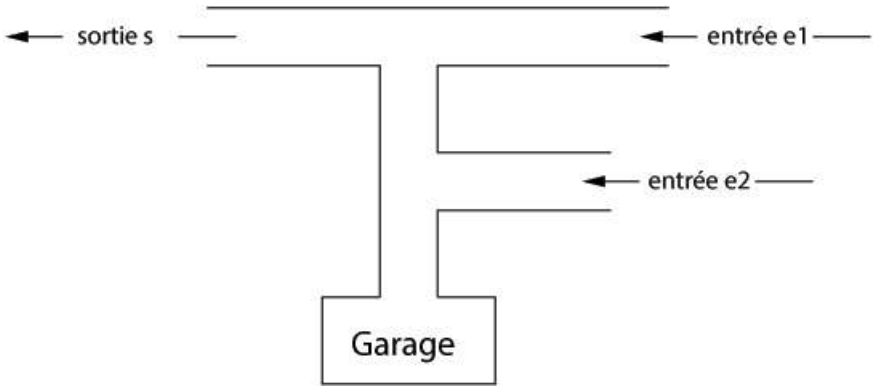

FIGURE 1 – garage à deux entrées et une sortie.

1. Est-ce que les séquences d'entrées/sorties suivantes de véhicules sont valides ?
   (a) $(v_1, e_1),(v_2, e_1),(v_3, e_1),s,s,(v_4, e_2),(v_5, e_1),s,s,s,(v_6, e_2),s$
   (b) $(v_1, e_1),(v_2, e_2),s,(v_3, e_2),s,s,s,(v_4, e_1),(v_5, e_2),s,(v_6, e_1),(v_7, e_2),s,s$
   Dans les cas où la séquence n'est pas valide, indiquer brièvement pourquoi.

2. On peut symboliser l'action "entrer une voiture" par les symboles $E1$ ou $E2$ (*empiler*) suivant l'entrée empruntée et l'action "sortir une voiture" par le symbole D (*dépiler*).
   La suite d'actions présentée au début peut alors être symbolisée par la séquence suivante :

   $$E1E1E2E1DDE2E2DDDE1E2E2DDDD$$

   Donner une règle qui caractériserait les séquences admissibles.

Exercice 2 (Dichotomie : "chemin" de recherche – *2 points*)

Supposons des listes d'entiers triées en ordre croissant. Si on effectuait dans celles-ci une recherche dichotomique de la valeur 66 : Parmi les séquences suivantes, lesquelles **ne pourraient pas** correspondre à la suite des valeurs rencontrées lors de la recherche ? Entourer sur les feuilles de réponses les numéros de séquences impossibles.

① 46 - 65 - 81 - 73 - 70 - 66
② 31 - 62 - 90 - 72 - 61 - 66
③ 36 - 70 - 53 - 50 - 61 - 66
④ 35 - 51 - 55 - 58 - 61 - 66

Python : procédé de Kaprekar

Les exercices de cette partie sont indépendants, SAUF le dernier exercice qui utilise les fonctions des exercices précédents.
Pour toutes les fonctions, on supposera que les paramètres sont valides : il n'y a pas de test à faire, pas d'exception à déclencher.

Exercice 3 (Test - *1 point*)

Soit la fonction `test` suivante :

```
1  def test(x, L):
2      i = len(L) - 1
3      while i >= 0 and L[i] != x:
4          i = i - 1
5      return i >= 0
```

Que fait cette fonction ?

Exercice 4 (Entiers ↔ liste – *5 points*)

1. Écrire la fonction `int_to_list(`n`, `p`)` qui convertit le nombre n (entier naturel à au plus p chiffres) en une liste de ses chiffres éventuellement complétés par des `0` pour atteindre p chiffres.

   Exemples d'applications (l'ordre des chiffres dans le résultat importe peu) :

   ```
   >>> int_to_list(27972, 5)
   [2, 7, 9, 7, 2]

   >>> int_to_list(42, 5)
   [2, 4, 0, 0, 0]
   ```

2. Écrire la fonction `list_to_ints(`L`)` qui, à partir de la liste L, ne contenant que des chiffres (de 0 à 9), retourne le couple $(left, right)$, avec
   — $left$: le nombre construit avec les chiffres de L "lus" de gauche à droite,
   — $right$: le nombre construit avec les chiffres de L "lus" de droite à gauche.

   Exemples d'applications :

   ```
   >>> list_to_ints([1,2,3,4,5,6])
   (123456, 654321)

   >>> list_to_ints([2,4,0,0,0])
   (24000, 42)
   ```

Exercice 5 (Histogramme et tri – *4 points*)

Nous travaillons ici sur des listes qui ne contiennent que des chiffres (de 0 à 9).

1. Écrire la fonction `hist(`L`)` qui retourne un histogramme (sous la forme d'une liste) des chiffres présents dans la liste L.

   Rappel : l'histogramme H est une liste telle que $H[i]$ est le nombre d'occurrences de la valeur i. Par exemple dans la première application ci-dessous, il y a 5 valeurs 0, 3 valeurs 1, 3 valeurs 2....

   Exemples d'applications :

```
>>> hist([1,5,9,3,0,1,2,0,1,0,2,5,0,5,2,0])
[5, 3, 3, 1, 0, 3, 0, 0, 0, 1]
>>> hist([1,0,1,0,1,0,1,0,1])
[4, 5, 0, 0, 0, 0, 0, 0, 0, 0]
```

2. Utiliser la fonction `hist` pour écrire la fonction `sort(`L`)` qui trie la liste L en ordre croissant (la fonction construit une nouvelle liste qui doit être retournée).

   Exemple d'application :

```
>>> sort([1,5,9,3,0,1,2,0,1,0,2,5,0,5,2,0])
[0, 0, 0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 5, 5, 5, 9]
```

Exercice 6 (Kaprekar – *5 points*)

Procédé de Kaprekar :

Soit un nombre entier n à p chiffres. Par exemple $n = 6264$.

— Prendre les p chiffres de n pour former deux nombres : le plus grand et le plus petit : 6642 et 2466.

— Soustraire en complétant éventuellement par des 0 à gauche pour obtenir à nouveau un nombre à p chiffres : n = 6642 - 2466 = 4176.

— Recommencer le procédé avec le résultat. 4176 → 7641 - 1467 = 6174

Le procédé de Kaprekar peut être utilisé avec un nombre quelconque. Dans tous les cas, quelque soit le nombre de chiffres, on arrivera à une valeur déjà rencontrée (dans les exemples donnés plus loin : on retrouve 6174 dans le premier, et 63 dans le deuxième).

Ce procédé peut être décrit par le "pseudo-algorithme" suivant :

```
procedure Kaprekar(entier n, p)

variables
   entier   low, high
   liste L
debut
   L ← liste-vide
   tant que n ∉ L faire
      ajouter x à L
      low ← nombre constitué des chiffres de n en ordre croissant
      high ← nombre constitué des chiffres de n en ordre décroissant
      n ← high - low
   fin tant que
fin
```

La fonction à écrire `Kaprekar(`n`,` p`)` prend en paramètre un entier naturel n et son nombre de chiffres p et applique le procédé de Kaprekar. De plus elle affiche les différentes valeurs calculées, comme dans les exemples suivants.

Exemples d'applications :

```
>>> Kaprekar(1574, 4)
1574 -> 6084 -> 8172 -> 7443 -> 3996 -> 6264 -> 4176 -> 6174 -> 6174

>>> Kaprekar(42, 2)
42 -> 18 -> 63 -> 27 -> 45 -> 9 -> 81 -> 63
```

Vous pouvez utiliser les fonctions des exercices 3 à 5 même si elles ne sont pas écrites.

Annexe : Fonctions et méthodes autorisées

Vous pouvez utiliser la méthode `append` et la fonction `len` sur les listes :

```
>>> help(list.append)
Help on method_descriptor:    append(...)
    L.append(object) -> None -- append object to end of L

>>> help(len)
Help on built-in function len in module builtins:    len(...)
    len(object)
    Return the number of items of a sequence or collection.
```

Vous pouvez également utiliser les fonctions `range` et `print`. Rappels :

```
>>> for i in range(10):
...     print(i, end=' ')
0 1 2 3 4 5 6 7 8 9

>>> for i in range(5, 10):
...     print(i, end='-')
5-6-7-8-9-
```