Algorithmique
Partiel n° 1 (P1)

Info-sup S1
Epita

18 janvier 2021 - 8h30

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

Exercice 1 (Double 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* e_1 et *entrée* e_2) et pour la ressortir une seule voie (*sortie* s).

Les mouvements possibles des voitures sont :

— une voiture peut entrer par l'*entrée* e_1
— une voiture peut entrer par l'*entrée* e_2
— 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

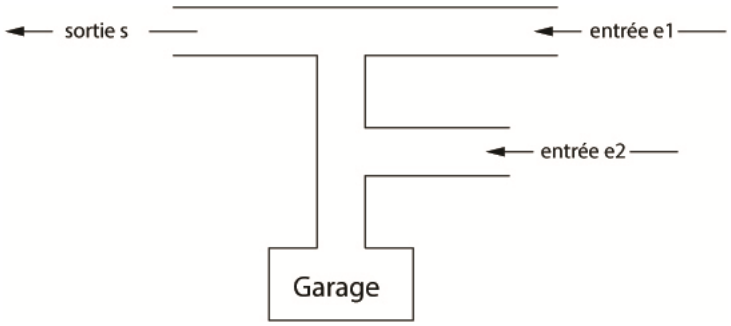

FIGURE 1 – garage à deux entrées et une sortie

Remarques :
— Il n'y a pas de problème de voitures se retrouvant face à face.
— Le garage est vide au départ et doit l'être à la fin.

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.

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. Donner une règle générale qui caractériserait les séquences admissibles.

Exercice 2 (Dichotomie – *3 points*)

On considèrera une version de l'algorithme de recherche dichotomique vu en cours qui s'arrête dès que les bornes se croisent ou sont identiques.

1. Compléter l'arbre de décision de la recherche dichotomique sur une liste de 16 éléments. Chaque noeud représente l'intervalle de recherche (les bornes gauche et droite) ainsi que l'indice calculé du milieu.
2. (a) Soit une liste de 32768 éléments triés en ordre croissant. Combien de comparaisons d'éléments seront faites, au pire des cas, dans le cas d'une recherche négative (réponse entière) ?
   (b) Soit k la réponse à la question précédente. Quelle peut-être, au maximum, la longueur de la liste qui générera $k + 2$ comparaisons lors d'une recherche négative ?

Exercice 3 (Recherche des indices – *3,5 points*)

Écrire la fonction `search_indexes(L, val)` qui retourne les valeurs du premier et du dernier indices de l'élément *val* dans la liste triée en ordre croissant L.

Si l'élément *val* n'est pas dans la liste, la fonction retourne le couple `(-1, -1)`.

Exemples d'applications :

```
>>> search_indexes([1, 4, 12, 12, 42], 4)
(1, 1)

>>> search_indexes([1, 4, 12, 12, 42], 15)
(-1, -1)

>>> search_indexes([1, 4, 12, 12, 42], 12)
(2, 3)

>>> search_indexes([4, 4, 4], 4)
(0, 2)
```

Exercice 4 (Somme consécutive – *4,5 points*)

Écrire la fonction `consecutive_sum(`L`, `S`)` qui vérifie s'il existe une suite d'éléments consécutifs dont la somme est S (avec $S > 0$) dans la liste L. La liste L ne contient que des entiers positifs ou nuls.

Exemples d'applications :

```
>>> consecutive_sum([3, 1, 4, 2, 4], 10)
True

>>> consecutive_sum([3, 7, 6], 6)
True

>>> consecutive_sum([], 6)
False

>>> consecutive_sum([3, 7, 6], 9)
False
```

Exercice 5 (Séparation – *6 points*)

Écrire la fonction `separate(`$L1$`, `$L2$`, `n`)` qui prend en paramètre deux listes triées en ordre croissant $L1$ et $L2$ et un entier n, et qui retourne un couple de listes dont :

— la première liste contient les éléments plus petits que n

— la seconde liste contient les éléments plus grands ou égaux que n

Ces deux listes résultats sont triées en ordre croissant.

Exemples d'applications :

```
>>> separate([1, 3, 12, 31, 42], [5, 16, 28], 12)
([1, 3, 5], [12, 16, 28, 31, 42])

>>> separate([1, 3, 12, 31, 42], [5, 16, 28], 42)
([1, 3, 5, 12, 16, 28, 31], [42])

>>> separate([1, 2, 3], [2, 5, 12, 42], 1)
([], [1, 2, 2, 3, 5, 12, 42])

>>> separate([],[2, 5, 12, 42], 6)
([2, 5], [12, 42])
```

Annexe : Fonctions et méthodes autorisées

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

```
>>> L = []

>>> for i in range(5):
        L.append(i)

>>> L
[0, 1, 2, 3, 4]

>>> len(L)
5

>>> for i in range(5, 10):
    L.append(i)
>>> L
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Aucun opérateur n'est autorisé sur les listes (`+`, `*`, `==` ...).

Vos fonctions

Vous pouvez également écrire vos propres fonctions, dans ce cas vous devez donnez leurs spécifications : on doit savoir ce qu'elles font.

Dans tous les cas, la dernière fonction écrite doit être celle qui répond à la question.