Le JEU pour REVISER les bases du PYTHON

Récapitulatif des cours présents dans le jeu • **Chapitre 5**

Table des matières

Les algorithmes de Première

Partie 1

Les **algorithmes** sont des ensembles d'instructions composants les programmes informatiques.

Certains algorithmes ont été étudiés et travaillés pour répondre à des problématiques précises. En Première, vous avez dû voir les 5 algorithmes suivants :

- Le tri par **sélection** d'une liste
- Le tri par **insertion** d'une liste
- La recherche **dichotomique** dans une liste
- L'algorithme **glouton**
- L'algorithme **k plus proches voisins**

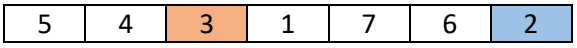
1. Les algorithmes de tri

Les algorithmes de tri sont un passage obligé en programmation. Il en existe des très efficaces[1] et des peu efficaces, voire absurdes[2].

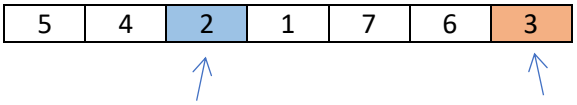
A. L'échange de deux éléments dans une liste

Un algorithme qui sera utile pour les algorithmes de tri quand nous allons étudiés est l'échange de 2 éléments dans une liste.

Prenons une liste d'entiers quelconques :



Le principe est d'échanger la place de 2 éléments entre eux :



[1] Le tri fusion ou le tri rapide (le tri fusion est au programme de Terminale)

[2] Le tri stupide, ou bogo-tri, consiste à mélanger la liste aléatoirement jusqu'à espérer obtenir une version triée (elle est particulièrement inefficace !)

En Python, il y a deux façons d'écrire un échange d'éléments à la position x et y dans une liste L :

```
temporaire = L[x]
L[x] = L[y]
L[y] = temporaire

# L'échange ne fonctionne pas avec
# le code suivant :
# L[x] = L[y]
# L[y] = L[x]
# d'où la présence de la
# variable 'temporaire'
```

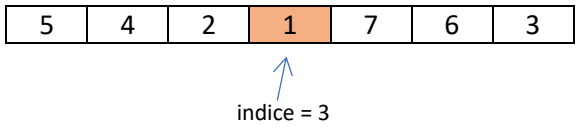
```
L[x], L[y] = L[y], L[x]

# Cette version n'est pas
# utilisable avec tous
# les langages
```

B. Trouver l'élément minimal d'une liste

Le parcours d'une liste pour trouver l'élément le plus petit est également un classique en programmation. Cet algorithme nous sera utile pour le tri sélection.

Ici, on s'intéressera à retrouver l'indice du plus petit élément d'une partie de la liste.



L'écriture Python de la recherche de l'indice du minimum entre les indices x (inclus) et y (exclus) de liste L :

```
indice_min = x
for j in range(x, y):
    if L[j] < L[indice_min]:    # Si un elm est plus petit que celui
        indice_min = j          # dont l'indice est stocké dans
                                # 'indice_min', on prend l'indice de
                                # cet elm comme indice d'élément minimal
```

Il est aussi possible d'écrire cet algorithme avec une boucle ***while****.*

Illustration de l'algorithme : *Initialisations : x = 1 et y = 6 donc indice_min = 1*

indices	0	1	2	3	4	5	6
	5	4	2	1	7	6	3
Etape 1	5	4	2	1	7	6	3
Etape 2	5	4	2	1	7	6	3
Etape 3	5	4	2	1	7	6	3
Etape 4	5	4	2	1	7	6	3
Etape 5	5	4	2	1	7	6	3

indice_min	if	j
1	False	1
2	True	2
3	True	3
3	False	4
3	False	5

C. Le tri sélection

Imaginons un paquet de cartes :

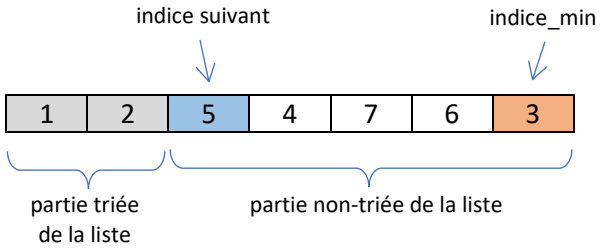
- à **droite** vous avez le paquet **pas trié**
- à **gauche** vous voulez avoir le paquet **trié**

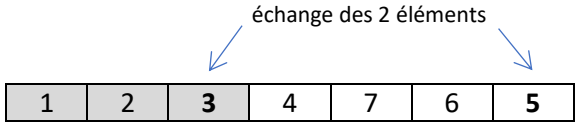
Le but est de prendre la carte la plus petite du paquet à droite, puis de la placer dans le paquet de gauche.

Exemple :

	Paquet trié	Paquet pas trié
Etape 1	Et on le place à gauche	5 4 2 **1** 7 6 3 — On prend l'élément le plus petit
Etape 2	1	5 4 **2** 7 6 3
Etape 3	1 2	5 4 7 6 **3**
Etape 4	1 2 3	5 **4** 7 6
etc.		

C'est exactement comme ça que l'algorithme **tri sélection** fonctionne !





Nous allons avoir besoin **d'échanger des éléments** dans une liste et de **trouver un élément minimal** dans une sous-liste.

Pratique ! On a justement vu comme faire dans les parties précédentes !

Proposition de code Python pour le tri sélection de la liste L :

```
n = len(L)

for i in range(0, n-1):        # Parcours de la liste, sauf le dernier elm
    indice_mini = i
    for j in range(i, n) :  # Parcours de la partie non-triée
        if L[j] < L[indice_mini]:
            indice_mini = j

    L[i], L[indice_mini] = L[indice_mini], L[i]
```

Trouver l'indice du minimum

échange

Illustration de l'algorithme :

indices	0	1	2	3	4	5	6
	5	4	2	1	7	6	3
Etape 1	5	4	2	1	7	6	3
	5	4	2	1	7	6	3
	5	4	2	1	7	6	3
	5	4	2	1	7	6	3
	5	4	2	1	7	6	3
	5	4	2	1	7	6	3
Etape 2	1	4	2	5	7	6	3
	1	4	2	5	7	6	3
	1	4	2	5	7	6	3
	1	4	2	5	7	6	3
	1	4	2	5	7	6	3
	1	4	2	5	7	6	3
Etape 3	1	2	4	5	7	6	3
	1	2	4	5	7	6	3
	1	2	4	5	7	6	3
	1	2	4	5	7	6	3
	1	2	4	5	7	6	3
Etape 4	1	2	3	5	7	6	4
	1	2	3	5	7	6	4
	1	2	3	5	7	6	4
	1	2	3	5	7	6	4
Etape 5	1	2	3	4	7	6	5
	1	2	3	4	7	6	5
	1	2	3	4	7	6	5
Etape 6	1	2	3	4	5	6	7
	1	2	3	4	5	6	7
FIN	1	2	3	4	5	6	7

indice_min	i	j
0	0	0
1		1
2		2
3		3
3		4
3		5
3		6
1	1	1
2		2
2		3
2		4
2		5
2		6
2	2	2
2		3
2		4
2		5
6		6
3	3	3
3		4
3		5
6		6
4	4	4
5		
6		6
5	5	5
5		6

D. Le tri par insertion

Imaginons un paquet de cartes :

- à **droite** vous avez le paquet **pas trié**
- à **gauche** vous voulez avoir le paquet **trié**

Le but est de prendre la première carte du paquet à droite, puis de la placer à la bonne place dans le paquet de gauche.

Exemple :

	Paquet trié	Paquet pas trié
Etape 1	On garde la première carte 5 On le place à la bonne place (pour garder le paquet trié)	4 2 1 7 6 3 On prend le premier élément
Etape 2	4 5	2 1 7 6 3
Etape 3	2 4 5	1 7 6 3
Etape 4	1 2 4 5	7 6 3
etc.		

Nous allons avoir besoin de **placer un élément** dans une sous-liste triée. Pour cela, on aura besoin **d'échanger des éléments** dans cette sous-liste.

Proposition de code Python pour le tri par sélection de la liste **L** :

```
for i in range(1, len(liste)):      # Parcours à partir de l'indice 1
    j = i

    # Parcours de la sous-liste triée (de j à 0) tant que l'élément précédent
    # est plus grand (j-1) que l'élément à placer (j).
    while liste[j-1] > liste[j] and j > 0:
        liste[j-1], liste[j] = liste[j], liste[j-1]      } échange
        j = j-1
```

Illustration de l'algorithme :

	indices	0	1	2	3	4	5	6	i	j
		5	4	2	1	7	6	3		
Etape 1		5	4	2	1	7	6	3	1	1
		4	5	2	1	7	6	3		0
Etape 2		4	5	2	1	7	6	3	2	2
		4	2	5	1	7	6	3		1
		2	4	5	1	7	6	3		0
Etape 3		2	4	5	1	7	6	3	3	3
		2	4	1	5	7	6	3		2
		2	1	4	5	7	6	3		1
		1	2	4	5	7	6	3		0
Etape 4		1	2	4	5	7	6	3	4	4
		1	2	4	5	7	6	3		4
Etape 5		1	2	4	5	7	6	3	5	5
		1	2	4	5	6	7	3		4
Etape 6		1	2	4	5	6	7	3	5	6
		1	2	4	5	6	3	7		5
		1	2	4	5	3	6	7		4
		1	2	4	3	5	6	7		3
		1	2	3	4	5	6	7		2
	FIN .	1	2	3	4	5	6	7		

E. Conseil

Il est conseillé d'utiliser une copie de la liste si l'on place ces algorithmes dans une fonction.

En effet, ces algorithmes modifient les listes utilisées.

Voire le chapitre précédent pour plus de détails.

2. Les algorithmes de recherche

Les algorithmes de recherche sont eux aussi un incontournable de la programmation. Ils consistent à retrouver un élément dans une liste.

L'algorithme qui nous vient tout de suite en tête est un simple parcours de la liste. A la fin du parcours, on sait si l'on a rencontré l'élément ou non.

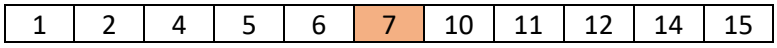
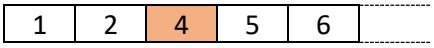
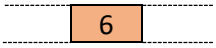
C'est ce qu'on appelle un algorithme « naïf » puisqu'il existe des algorithmes plus optimisés (avec une complexité plus faible) pour résoudre ce problème.

L'**algorithme dichotomique** est l'un d'eux. **/!** il ne fonctionne qu'avec une liste triée

Connaissez-vous **le jeu du juste prix**[3] ?

L'algorithme dichotomique fonctionne avec le même principe.

Exemple : Je cherche si le chiffre **6** est dans la liste triée

Etape 1	 6 est plus petit que 7 On cherche dans la sous-liste à gauche de 7
Etape 2	 6 est plus grand que 4 On cherche dans la sous-liste à droite de 4
Etape 3	 6 est plus grand que 5 On cherche dans la sous-liste droite
Etape 4	 Trouvé !

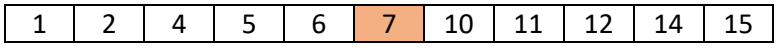
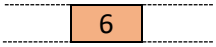
[3] Deviner un nombre aléatoire. A chaque tentative, on indique si le nombre à trouver est plus grand ou plus petit. Il est généralement conseillé de donner un nombre au milieu des bornes les plus proches connues. Exemple : je sais que le nombre est entre 20 et 100, je propose « 60 ». Si le nombre est plus grand, il est entre 60 et 100, je propose « 80 » et ainsi de suite.

A. Rappel : Calculer le milieu de deux indices

Faisons un peu de math !

Comment vous calculer votre moyenne si vous avez eu 13 et 16 à vos devoirs ? Alors vous savez calculer un milieu :)

Reprenons notre exemple :

Etape 1	 On cherche dans toute la liste : de l'indice 0 à 10 (0 + 10) / 2 = **5** **7** est à l'indice **5**
Etape 2	 On cherche dans toute la liste : de l'indice 0 à **5-1** (0 + 4) / 2 = **2** **4** est à l'indice **2**
Etape 3	 On cherche dans toute la liste : de l'indice **2+1** à 4 (3 + 4) / 2 = 3,5 => on prend la partie entière, soit **3** **5** est à l'indice **3**
Etape 4	 On cherche dans toute la liste : de l'indice **3+1** à 4 (4 + 4) / 2 = **4** **6** est à l'indice **4**

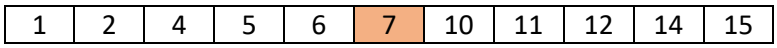
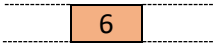
Pour être sûr d'avoir un entier lors de la division, il faut utiliser des doubles slash en Python.

Exemple : `(3 + 4) // 2 = 3`

B. Rappel : Sélectionner une partie d'une liste

Il peut vous être utiles de sélectionner une partie de la liste. Pour cela, Python a des instructions pour cela :

Reprenons notre exemple : Notons **L** la liste utilisée en exemple

Etape 1	 On cherche dans toute la liste : de l'indice 0 à 10 On prend L au complet
Etape 2	 On cherche dans toute la liste : de l'indice 0 à **5-1** `L[:5]` ou `L[0:5]` (5 non inclus)
Etape 3	 On cherche dans toute la liste : de l'indice **2+1** à 4 `L[3:5]` (5 non inclus)
Etape 4	 On cherche dans toute la liste : de l'indice **3+1** à 4 `L[4]` (**/!** `L[4:4]` donne `[]`)

C. L'algorithme dichotomie

Voici plusieurs versions de cet algorithme en Python.

Il en existe d'autres (comme tous les algorithmes), mais chaque version proposée ci-dessous propose des particularités intéressantes à connaître (à mon sens).

Sans modifier la liste :

```
def dichotomie(L, nombre):
      # Calcul du milieu
      i_min = 0
      i_max = len(L)-1
      milieu = (i_min + i_max) // 2

      while len(L[i_min:i_max+1]) > 1 and L[milieu] != nombre:
      # ou
      while i_min < i_max and L[milieu] != nombre:

            if L[milieu] < nombre:
                  # Calcul du nouveau milieu
                  i_min = milieu+1
                  milieu = (i_min + i_max) // 2

            else:
                  # Calcul du nouveau milieu
                  i_max = milieu-1
                  milieu = (i_min + i_max) // 2


      return L[milieu] == nombre
      # ou
      if L[milieu] == nombre:
            return True
      else:
            return False
```

Avec modification la liste : **/!** il faut copier la liste pour ne pas la modifier dans le programme principal

```
from copy import copy

def dichotomie(liste, nombre):
      L = copy(liste)
      milieu = len(L) // 2
```

```
while L != []:

      if L[milieu] == nombre:
            return True

      elif L[milieu] < nombre:
            L = L[milieu+1:]
            milieu = len(L) // 2

      else:
            L = L[:milieu]
            milieu = len(L) // 2

return False
```

identiques

```
# ou

while len(L) > 1 and L[milieu] != nombre:

      if L[milieu] < nombre:
            L = L[milieu+1:]
            milieu = len(L) // 2

      else:
            L = L[:milieu]
            milieu = len(L) // 2

return L[milieu] == nombre
```