Algorithmes de tri

Table des matières

Dans ce qui suit, **L** est une liste contenant n éléments qui peuvent être des entiers, des réels ou, plus généralement, toute collection de valeurs sur lesquelles on peut définir une relation d'ordre total.

On convient que :

- $\mathbf{L}[i]$ désigne l'élément d'indice i de **L**. $\mathbf{L}[0]$ est le premier élément et $\mathbf{L}[n-1]$ le dernier.
- $\mathbf{L}[i,j]$ désigne la sous-liste composée des éléments allant de $\mathbf{L}[i]$ à $\mathbf{L}[j]$ **inclus**.

On se propose dans ce chapitre d'étudier différents algorithmes qui permettent de trier les différents éléments de **L** en les rangeant du plus petit, placé dans $\mathbf{L}[0]$, au plus grand, placé dans $\mathbf{L}[n-1]$.

1 Tri par insertion

1.1 Principe

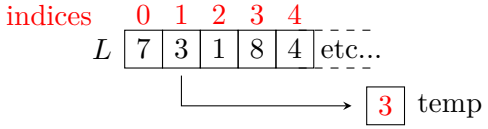
Cet algorithme s'inspire du tri que fait un joueur de cartes lors de la distribution du jeu. Les cartes qu'il a en main sont déjà triées, la plus petite étant à gauche et la plus grande à droite. Lorsqu'on lui distribue une nouvelle carte, il va l'insérer à sa place en la comparant aux plus grandes valeurs de son jeu.

Schéma :

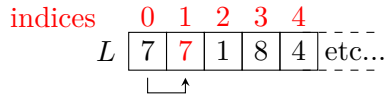
Prenons l'exemple de la liste $L = [7, 3, 1, 8, 4, 5, 12]$.

Début :

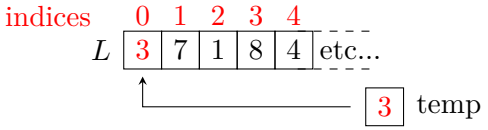
- On crée un trou en sauvegardant la valeur dans une variable `temp` :



- On décale les éléments vers la droite ... ce qui revient à décaler le trou vers la gauche :

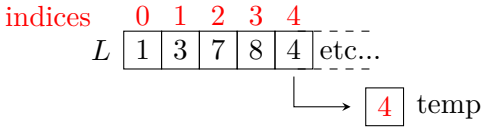


- On remet l'élément à sa bonne place, puis on passe à l'indice suivant :

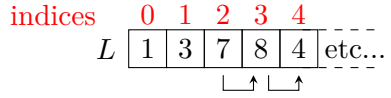


Au cours de l'algorithme :

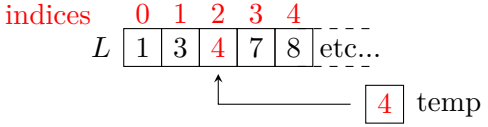
- On crée un trou :



- On décale le trou vers la gauche :



- On remet l'élément à sa bonne place, puis on passe à l'indice suivant :




```
def Tri_Insertion(L) :
    n = len(L)          # On récupère la taille de la liste
    for i in range(1,n) :
        temp = L[i]          # On crée un trou
        j = i          # j est l'indice du trou
        while ( j > 0 and L[j − 1] > temp ) :
            L[j] = L[j − 1]          # décalage du trou à gauche
            j = j − 1
        L[j] = temp          # On remet l'élément à sa bonne place
```

Commentaires :

- On commence au second élément de L : $i = 1$ et non $i = 0$
- Dans la condition de la boucle **while** il faut mettre $j > 0$ en premier car l'interpréteur de python commence par évaluer la véracité de la première partie de l'expression. Ainsi, si $j \leqslant 0$, toute l'expression est **False** et python ne va pas lire la seconde partie : $L[j-1] >$ temp qui provoquerait un débordement d'indice !
- j est l'indice du trou.
- Il n'y a pas de **return**. En python, on peut modifier dans un fonction une liste passée en paramètre. L sera donc bien modiée par ce code.
- Vous trouverez une animation de cette fonction sur le site MP1 dans le fichier **insertion.odp** (diaporama).

1.2 Mise en œuvre

1. Commençer par créer une liste de n nombres entiers aléatoires compris entre 0 et 100. Pour cela, importer la fonction `randint` du module `random` grâce à l'instruction :

```
from random import randint
```

L'appel de `randint(a,b)` où a et b sont deux entiers ($a < b$) renvoie un entier aléatoire m vérifiant $a \leqslant m \leqslant b$.

2. Écrire une fonction `listeEntiers(n)` qui renvoie une liste de n entiers choisis aléatoirement dans l'intervalle entier $[\![0, 100]\!]$.
3. Écrire l'algorithme de tri par insertion et le tester avec une liste renvoyée par `listeEntiers(n)`.

Vous trouverez la correction sur le fichier `CorrigeTris.py` sur le site. Entraînez-vous à écrire ces codes.

2 Tri fusion

2.1 Principe

Prenons l'exemple d'un jeu de 32 cartes à trier.

- L'idée est de partager ce jeu en deux jeux de 16 cartes et de trier séparément ces deux jeux.
- On ré-assemble ensuite chacun des deux jeux, grâce à un ***algorithme de fusion***.

Le tri fusion fonctionne de manière **récursive** :

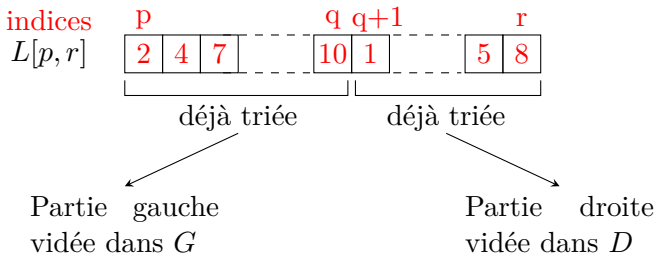
- Chaque jeu de 16 cartes est divisé en 2 jeux de 8 cartes qui sont triés séparément puis assemblés dans le bon ordre au moyen de l'algorithme de fusion.
- Chacun des jeux de 8 cartes aura été au préalable séparé en 2 jeux de 4 cartes qui auront été triés puis ré-assemblés dans le bon ordre.
- etc...

2.2 Algorithme de fusion

L'**algorithme de fusion** est le *cœur du tri fusion*. C'est lui qui est chargé de ré-assembler deux listes triées séparément pour n'en former qu'une dans laquelle tous les entiers sont triés.

Schéma :

- On commence par s'intéresser à une sous-liste $L[p, r]$, $p < r$ de L, elle-même divisée en deux sous-listes adjacentes $L[p, q]$ et $L[q+1, r]$ que l'on suppose *déja triées*. On a donc, par exemple :



- On remet ensuite un à un les éléments de G et de D dans $L[p, r]$ en les classant dans le bon ordre.
- Pour y arriver sans vérifier à chaque fois si on a atteint ou non la fin de G ou D, on place une **butée** dans chacune de ces listes. Cette butée est appelée infini dans le programme suivant : il s'agit d'un entier strictement supérieur à tous les entiers de la liste L.

```
def Fusion(L, p, q, r) :
    G, D = [], []
    for i in range(p, q + 1) :          # i va de p à q
        G.append(L[i])
    for j in range(q + 1, r + 1) :      # j va de q+1 à r
        D.append(L[j])
    infini = max(L)+1    # butée = élémenr > tous les éléments de L
    G.append(infini)
    D.append(infini)
    i, j = 0, 0    # i indice dans G, j indice dans D
    for k in range(p,r + 1) :
        if G[i] <= D[j] :
            L[k] = G[i]
            i = i + 1
        else :
            L[k] = D[j]
            j = j + 1
```

Commentaires :

- i est l'indice courant de G, j celui dans D et k est l'indice dans L lorsqu'on remet les éléments en place dans le bon ordre.
- Vous trouverez une animation de cette fonction sur le site MP1 dans le fichier **fusion.odp** (diaporama).

2.3 Algorithme du tri par fusion

Soit à trier la sous-liste $\mathbf{L}[p,\, r]$ $(p \leqslant r)$ extraite de $\mathbf{L}$.

La fonction `Tri_Fusion` prend en paramètres la liste $\mathbf{L}$ et les deux entiers p et r.

- Elle divise $\mathrm{L}[p,\, r]$ en deux sous-listes de tailles à peu près identiques ...
- ... qu'elle trie séparément, avec une **approche complètement récursive**...
- ... avant de fusionner les deux sous listes qui viennent d'être triées.

```
def Tri_Fusion(L, p, r) :
    if p < r :
        q = int( (p + r)/2 )
        Tri_Fusion(L, p, q)
        Tri_Fusion(L, q + 1, r)
        Fusion(L, p, q, r)
```

Commentaires :

- On teste si $p < r$. Dans le cas contraire, on a soit une absurdité $(r < p)$, soit il n'y a rien à trier $(p == r)$ puisque la sous-liste $L[p, r]$ ne renferme alors qu'un seul élément. Cette ligne sert aussi de condition de terminaison des appels récursifs.
- On définit ensuite l'entier $q = \mathrm{E}\left(\dfrac{p+q}{2}\right)$ (partie entière) ce qui permet de diviser $\mathrm{L}[p, r]$ en deux sous-listes de tailles à peu près égales : $\mathrm{L}[p, q]$ et $\mathrm{L}[q + 1, r]$.
- La fonction s'appelle ensuite de façon récursive pour trier les deux sous-liste obtenues.
- On ré-assemble enfin les deux sous-listes triées grâce à la fonction `Fusion` étudiée à la section précédente.

- La procédure de tri de la liste entière L est lancée en appelant la fonction `Tri_Fusion`(L, 0, $n-1$).

2.4 Mise en œuvre

Écrire l'algorithme de tri fusion et le tester avec une liste renvoyée par `listeEntiers(n)`.

2.5 Application : recherche de la médiane dans une liste

Soit $L = [x_0, ..., x_{n-1}]$ une liste de n valeurs distinctes deux à deux et appartenant à un ensemble totalement ordonné. On lui associe la liste triée $\hat{L} = [\hat{x}_0, ..., \hat{x}_{n-1}]$ formée des mêmes éléments écrits dans l'ordre croissant.

- Lorsque $n = 2p+1$ (entier impair), la valeur médiane est la valeur $\hat{x}_p$ pour laquelle il existe p éléments strictement plus petits et p éléments strictement plus grands que $\hat{x}_p$.
- Lorsque $n = 2p$ (entier pair), on appelle valeurs médianes les deux éléments $\hat{x}_{p-1}$ et $\hat{x}_p$.

Réalisation :

1. Modifier la fonction `listeEntiers(n)` pour qu'elle ne contienne que des entiers distincts deux à deux.
2. Considérons une liste **L** non triée composée de n éléments tous distincts deux à deux. Écrire une fonction `mediane(L)` qui retourne la valeur ou les valeurs médianes de **L** en utilisant le tri fusion.
3. Tester cette fonction `mediane(L)` sur une liste renvoyée par `listeEntiers(n)`.