

TP 5 : Tris de listes

2 BCPST 2 - TP Informatique

CONSEIL : à chaque nouvelle question faites une nouvelle cellule en tapant "#%%" .
Vous pourrez ainsi traiter toutes les questions dans un même fichier.

Objectif du TP : - Connaître différentes méthodes de tri, à commencer par le tri par insertion.

I Tri par insertion

Le tri par insertion est sans doute le plus simple à comprendre. Le principe est de créer une liste triée `Lt` initialement vide et parcourir `L` pour ajouter ses éléments dans `Lt` un par un.

Exemple :

Q.1. Ecrire une fonction `insert_element(L,a)` qui prend en entrée une liste `L` triée et un élément `a` et qui retourne une liste triée contenant les éléments de `L` ainsi que `a`.

Exemple :

```
insert_element([1,5,6,9],7)
-----
-> [1,5,6,7,9]
```

Q.2. Ecrire une fonction `tri_insertion(L)` qui prend en entrée une liste `L` et qui retourne une liste contenant tous les éléments de `L` triés en suivant la méthode du tri par insertion.

II Tri à bulles

Le tri à bulles consiste à parcourir la liste entièrement plusieurs fois jusqu'à qu'elle soit triée. En parcourant la liste on regarde les éléments deux par deux. S'ils sont mal ordonnés, on les permute. Si la liste a été parcourue sans permutation, alors elle est triée et on s'arrête.

Par exemple pour la liste `L = [5,3,9,1]` :

On parcourt la liste une première fois :
`L = [ 5,3 ,9,1] → L = [ 3,5 ,9,1]`
`L = [3, 5,9 ,1] → L = [3, 5,9 ,1]`
`L = [3,5, 9,1 ] → L = [3,5, 1,9 ]`

La liste n'est pas triée, on recommence :
`L = [ 3,5 ,1,9] → L = [ 3,5 ,1,9]`
`L = [3, 5,1 ,9] → L = [3, 1,5 ,9]`
`L = [3,1, 5,9 ] → L = [3,1, 5,9 ]`

La liste n'est pas triée, on recommence :

`L = [ 3,1 ,5,9] → L = [ 1,3 ,5,9]`
`L = [1, 3,5 ,9] → L = [1, 3,5 ,9]`
`L = [1,3, 5,9 ] → L = [1,3, 5,9 ]`

Il faut recommencer alors une dernière fois pour constater qu'on n'effectue aucune permutation, ce qui veut dire que la liste est triée

Q.3. Ecrire une fonction `tri_bulle(L)` qui prend en entrée une liste `L` et qui retourne une liste contenant tous les éléments de `L` triés en suivant la méthode du tri à bulles.

III Tri par sélection

Il ressemble au tri par insertion.

Le tri par sélection consiste à aller chercher directement le minimum de la liste `L`, l'insérer dans la liste `Lt` triée qui est initialement vide et supprimer l'élément de `L`.

Q.4. Ecrire une fonction `minimum(L)` qui prend en entrée une liste `L` et qui retourne le minimum de la liste `L`.

Q.5. Ecrire une fonction `tri_selection(L)` qui prend en entrée une liste `L` et qui retourne une liste contenant tous les éléments de `L` triés en suivant la méthode du tri par sélection

IV Tri par comptage

Le tri par comptage se fait en maintenant une liste de compte `C`. Cet algorithme ne fonctionne que pour les listes `L` contenant des nombres entiers. Le principe est de chercher le min `a` et le max `b` de `L`.

On crée ensuite la liste `C` qui contient initialement `b-a+1` fois zéro. L'objectif de `C` est de compter combien de fois les éléments de `L` apparaissent. Ainsi `C[i]` contient le nombre d'apparition de l'élément `a+i` dans `L`.

On parcourt `L` une seule fois pour remplir `C`. Ensuite on parcourt `C` pour reconstruire `L` de manière triée.

Exemple avec `L = [5,3,9,1]` :

On a `a = 1` et `b = 9` donc `C` contient `b-a+1 = 9` éléments. Initialement `C = [0,0,0,0,0,0,0,0,0]`.
En parcourant `L`, on peut compter les éléments de `L` et rentrer les données dans `C` :
On trouve `C = [1,0,1,0,1,0,0,0,1]`
Par exemple `C[ 2 ] = 1` car `3 = 2 + 1 = 2 + a` apparaît 1 fois dans la liste `L`.

Q.6. Ecrire une fonction `liste_vers_C(L)` qui prend en entrée une liste `L` et qui retourne la liste de comptage `C` correspondante.

Q.7. Ecrire une fonction `tri_comptage(L)` qui prend en entrée une liste `L` et qui retourne une liste contenant tous les éléments de `L` triés en suivant la méthode du tri par comptage.

V Tri par fusion

Le tri par fusion utilise la méthode "diviser pour régner" c'est le plus efficace des tris présentés dans ce cours.

Q.8. Créer une fonction récursive `somme_terme_a_terme(L1,L2)` qui prend en entrée deux listes `L1` et `L2` de même taille et qui renvoie la liste contenant la somme des termes à terme des éléments de la liste.

Exemple :

```
L1 = [0,1,2,3,4]
L2 = [1,1,2,2,-1]
print(somme_terme_a_terme(L1,L2))
-----
-> [1,2,4,5,3]
```

Les fonctions suivantes ont pour objectif de créer un algorithme de tri :

Q.9. En vous inspirant des rappels de l'introduction, créer la fonction (non récursive) `separe(L)` qui sépare la liste `L` en deux listes de taille égale.

Exemple :

```
L = [0,1,2,3,4,5,6]
print(separe(L))
-----
-> ([0,1,2,3],[4,5,6])
```

Q.10. Créer la fonction récursive `fusion(L1,L2)` qui prend en entrée deux listes triées `L1` et `L2` et retourne la fusion triée des deux listes.

Q.11. Créer la fonction récursive `tri(L)` qui prend en entrée une liste `L` et retourne la version triée de la liste `L`.

Indication : On pourra utiliser les deux fonctions précédentes : on sépare la liste en deux, puis on trie chaque morceau puis on les fusionne ensemble pour obtenir la liste triée.

VI Exercice : Ordre alphabétique

Afin qu'il puisse les différencier, les caractères sont représentés au sein de l'ordinateur par des nombres. En Python, on peut obtenir le codage numérique d'un caractère avec la fonction `ord`.

Ainsi `ord('a')` vaut 97, `ord('b')` vaut 98, etc...

(Attention, les majuscules ont des valeurs numériques associées différentes)

On peut faire le codage inverse en utilisant la fonction `char`. Ainsi `char(97)` vaut `'a'`.

Q.12. En utilisant les fonctions ci-dessus, écrire une fonction `plus_grand_mot(s1,s2)` qui prend en entrée deux chaînes de caractères représentant des mots et qui retourne la chaîne la plus grande dans l'ordre alphabétique.

Q.13. Coder une fonction `tri_alphabétique(L)` qui prend en entrée une liste de mots `L` et qui retourne la liste de mots `L` triée par ordre alphabétique.