

TRIS DE LISTES

Pour trier une liste en Python il suffit de faire `>>> a=[3,2,1,5,8,1] >>> a.sort()` (pas d'affectation) `>>> a` `[1, 1, 2, 3, 5, 8]`

Noter qu'on peut aussi trier des chaînes de caractères

`>>> l=['bc','abc','aab','ba'], >>> l.sort()` `['aab', 'abc', 'ba', 'bc']` on peut ainsi faire facilement un programme de calcul de médiane `median(L)`:

```
L.sort()
N = len(L)
n = N//2.0
p = int(n)
if n == p:
    return ((L[p-1]+L[p])/2.0)
else:
    return (L[p])
```

Une liste L est indicée de 0 à n-1 ou `n=len(L)` longueur ;

L est modifiable on peut écrire `L[i]=` ;

On permute par `L[i], L[j] = L[j], L[i]`

`L[a :b]` donne `L[i]` de `i=a` à `b-1` ; `L[-1]` donne le dernier, `L[-2]` l'avant dernier

`range(a,b)` donne la liste de a compris à b non compris ; `range(n)` est la liste `[0,1,..., n-1]` ;

`range(a,b,pas)` donne `a,a+p,a+2p, .. a+kp <b`

Copie de la liste L : `L2=L[:]` ou `L2=list(L)` surtout pas `L2=L` car alors modifier L2 modifie L !

on ajoute x à la liste L avec `L.append(x)` ou bien `L=L+[x]` ; ajoute de deux listes : `L1.extend(L2)`

On enlève avec `L.remove(x)` (enlève le premier x trouvé de L) ou par `L.pop(i)` qui renvoie `L[i]` et l'enlève de la liste `L.pop(i)` enlève `L[i]`

(pas de return ou d'affectation sur les méthodes comme `append`, `remove`, `sort` .. si on le fait cela donne `None` soit rien)

Dans une fonction où la liste L est un paramètre python modifie l'original !! voir point important

Il y a plusieurs algorithmes de tris : tri par sélection , tri bulle , tri rapide

1/Ecrire un programme qui entre une liste L et qui donne un indice i_o du maximum et le maximum

2/Ecrire un programme qui entre L et qui place le maximum à la fin soit échange les indices i_{max} et $n-1$;

On peut écrire $L[i_{max}] , L[n-1] = L[n-1] , L[i_{max}]$

Point important : si on définit une liste a et on applique $p(a)$ et que le programme $p(L)$ modifie la liste L , à la sortie a sera modifié , python modifie l'original pour une liste ce qu'il ne fait pas pour un entier ou flottant ! si on veut garder le a de départ et obtenir une liste triée qui ne modifie pas l'original il faudra faire une copie de L et travailler sur la copie ; les fonctions python modifie l'original quand c'est une liste

3/Tri par sélection :

a) On trouve le maximum (entre 0 et $n-1$) et on le met à la fin $L[n-1]$;

Puis on cherche le maximum entre 0 et $n-2$ et on le met à l'avant dernière place $L[n-2]$

Etc...

On aura 2 boucles : boucle dans boucle

Remarque bis :

Si on ne veut pas modifier le paramètre L de départ il faut faire une copie et travailler sur la copie

b) Faire le tri par sélection en 2 modules : le premier programme $p_1(L,i)$ cherche l'indice du maximum de la liste entre 0 et $i-1$ et le permute avec $L[i-1]$

le deuxième programme fait le tri sélection

c)Complexité :

à l'étape 1 $n-1$ comparaisons et $n-1$ affectations au pire pour trouver le maximum (le pire cas est $12...n$) , puis $n-2$, puis $n-3$ au total $1+2+...+(n-1)=n(n-1)/2 = O(n^2)$ (grand O)

Ainsi doubler la taille de la liste , quadruple le temps d'exécution voir plus bas

d)Médiane : on fait l'étape 1,2 ,... jusqu'à l'étape $(n//2)+1$ et $L[n- n//2-1]$ donne la médiane

4/ Tri bulle :

a) Pour trouver le maximum on peut aussi permuter 2 éléments qui ne sont pas dans le bon ordre (l'ordre croissant) ; ainsi par permutation successive le maximum se retrouve à la dernière position

[5, 1, 4, 7, 2] → [1, 5, 4, 7, 2] → [1, 4, 5, 7, 2] → [1, 4, 5, 2, 7]

Programmer cette méthode pour trouver le maximum qui se retrouve en dernier

b) On n'a plus qu'à recommencer, pour que le « deuxième maximum » se retrouve en avant dernière place, etc...

c) complexité : au maximum $n-1$ permutations, $+n-2 + (n-3) + \dots + 1$ soit $n(n-1)/2$ c'est un $O(n^2)$

5/ Tri récursif : tri rapide

a) Ce tri en anglais quicksort date de 1960 inventé par Tony Hoare

Tri plus rapide en $n \ln(n)$ en moyenne : Pour $n=10^6$, $n^2=10^{12}$ mais $n \ln(n) \approx 10^7$

Double n , fait un peu plus que doubler le temps (rajoute un facteur en Cn négligeable par rapport à $n \ln(n)$ donc beaucoup mieux que les tris vus qui eux quadruple le temps si on double n)

L'idée est de séparer la liste initiale en 2 listes : L1 les éléments strictement inférieurs à un pivot par exemple $L[0]$ et L2 ceux strictement supérieurs au pivot.

Mais s'il y a plusieurs fois la valeur de pivot il y a un problème, pourquoi ?

On prendra donc supérieur ou égal au pivot mais **en supprimant** $L[0]$ (d'abord le stocker)

Ensuite on trie L1, L2 par récursivité puis on concatène L1, pivot, L2

Soit retourner : $L1 + [\text{pivot}] + L2$

Attention à l'initialisation L1 peut être vide (réfléchir au(x) condition(s) d'arrêt) ; en tout cas les longueurs de L1 et L2 sont bien de longueur strictement plus petite que celle de L et donc la récursivité peut s'appliquer.

pour trouver L1 on peut écrire $L1 = []$ for x in L if $x < \text{pivot}$: $L1.append(x)$

ou mieux liste en compréhension $L1 = [x \text{ for } x \text{ in } L \text{ if } x < \text{pivot}]$

pour L2 il ne faut pas reprendre le pivot $L[0]$! si on l'a éliminé il n'y a pas de problème

algorithme final :

$a = L[0]$

supprimer $L[0]$ de L

$L1$ est formé des éléments $< a$, $L2$ des éléments $\geq a$

Retourner $\text{tri}(L1) + [a] + \text{tri}(L2)$

b)Complexité : Si on considère qu'en moyenne on divise chaque liste en 2 il y aura k étapes avec

$n/2^k < 1$ Soit $\ln(n) / \ln(2)$ étapes ; dans une étape pour une liste de longueur k , on fait $k-1$ comparaisons , on compare chacun avec $L[1]$ de l'ordre de k soit de n

D'où $\ln(n)$ étape de n on a $n \ln(n)$ comme ordre de grandeur

Remarque :L'algorithme est lent lorsque le tableau est trié car si on partage avec $L[0]$, $L2$ est de longueur $n-1$, etc...Ce tri est donc à éviter dans une liste peu mélangée.

c)Tester la complexité avec le module time et le module random

from random import randint

randint(a,b) donne un entier au hasard compris entre a et b pouvant être a ou b

faire une fonction listhasard (n ,m) qui entre les entiers n , m et qui donne une liste de n entiers compris entre 0 et m :

on fera un certain nombre de fois (réfléchir au range) $x = \text{randint}(0,m)$ et on rajoute x à la liste

import time

$L = \text{listhasard}(400,1000)$

$t_0 = \text{time.clock}()$

$\text{trirapide}(L)$

print (time.clock()- t_0)

puis on recommence avec $\text{listhasard}(800,1000)$

Comparer aussi avec le tri selection ou bulle pour la rapidité et aussi la complexité du tri bulle ou par selection : si on passe de 400 à 800 cela devra quadrupler pour les tris non rapide et un peu plus que doubler pour le tri rapide

