Randomisation du quicksort

F. Kany. ISEN-Brest. La Croix-Rouge.

Présentation

L'algorithme de tri rapide (ou quicksort) est un algorithme de tri inventé par C.A.R. Hoare en 1960. Son principe est le suivant. Dans une liste de taille n, on choisit un élément p comme pivot. On partitionne la liste en deux parties : les éléments plus petits que p et les éléments plus grands que p.

Dans le cas idéal, la partition divise la liste initiale en deux listes de taille $n/2$.

En réitérant, on obtient des sous-listes de longueurs $n/4$, puis $n/8$, puis $n/16$,...

Au bout de k itérations, on obtient des listes de longueurs $n/2^k$. Lorsque les listes ne contiennent plus qu'un seul élément ($\frac{n}{2^k} = 1 \Rightarrow k = \frac{\ln n}{\ln 2} = \ln_2(n)$), le tri est fini. On a donc $k = \ln_2(n)$ itérations où, à chaque fois, il faut comparer tous les éléments à leurs pivots respectifs, soit n comparaisons. Dans le meilleur des cas, il faut donc $n.\ln_2(n)$ comparaisons. C'est la limite théorique minimale pour les algorithmes généraux de tri par comparaisons.

Sur le principe, le code est le suivant :

```
from sys import setrecursionlimit

setrecursionlimit(5000)

def tri(liste):
    if len(liste)>1:
        pivot = liste[0]
        petits = [v for v in liste if v<pivot]
        egaux = [v for v in liste if v==pivot]
        grands = [v for v in liste if v>pivot]
        return tri(petits)+egaux+tri(grands)
    else:
        return liste
```

Malheureusement, très souvent, on est amené à trier des listes qui sont déjà quasiment triées. La partition divise la liste initiale en deux listes : l'une de taille $n-1$ et l'autre de taille 1. Il faut donc n itérations pour que toutes les listes soient de taille 1. À chaque fois, il faut effectuer n comparaisons. Dans le pire des cas, il faut donc n^2 comparaisons. C'est équivalent aux moins bons algorithmes : tri à bulle, tri par insertion, tri par sélection,...

Une astuce consiste à prendre le pivot p au hasard dans la liste. Grossièrement, si on choisit p au hasard, on va avoir statiquement toutes les partitions de $(n-1, 1)$ à $(1, n-1)$. En moyenne, on aura des partitions de $(n/2, n/2)$.

Conclusion : même si la liste est déjà triée, la complexité reste de $n.\ln_2(n)$.

Question

1. Chronométrer cet algorithme quand on l'exécute sur une liste triée.

```
def test():
    liste=list(range(1000))
    tri(liste)

from timeit import timeit
print(timeit("test()",setup="from __main__ import test",number=100))
```

2. Randomiser l'algorithme en choisissant le pivot au hasard.
3. Re-chronométrer l'algorithme dans les mêmes conditions.