NSI · PREMIÈRE / COURS

Les algorithmes de tri

Astrolab Cours · astrolabcours.fr

01 Le problème

Trier 30 copies par note, c'est facile à la main. Mais comment le décrire à une machine, étape par étape, et combien de temps faudra-t-il pour un million de valeurs ?

01 L'idée : la main de cartes

Quand tu ramasses tes cartes une à une, tu glisses chaque nouvelle carte à sa place parmi celles déjà rangées dans ta main : c'est le **tri par insertion**. Une autre méthode : chercher la plus petite carte du paquet et la poser à gauche, puis la plus petite des restantes, et ainsi de suite : c'est le **tri par sélection**.

01 Le tri par sélection

Pour chaque position `i` de gauche à droite : chercher le **minimum** de `t[i:]` et l'**échanger** avec `t[i]`.

```
def tri_selection(t):
    n = len(t)
    for i in range(n - 1):
        imin = i
        for j in range(i + 1, n):
            if t[j] < t[imin]:
                imin = j
        t[i], t[imin] = t[imin], t[i]
```

Le tri par insertion

Pour chaque élément à partir du deuxième : le décaler vers la gauche tant que son voisin de gauche est plus grand.

```
def tri_insertion(t):
    for i in range(1, len(t)):
        x = t[i]
        j = i
        while j > 0 and t[j - 1] > x:
            t[j] = t[j - 1]
```

© Astrolab Cours · astrolabcours.fr/nsi/premiere/algorithmes-de-tri/ · Tous droits réservés : usage personnel uniquement, republication et usage commercial interdits

```
            j = j - 1
        t[j] = x
```

Pourquoi c'est juste : l'invariant

Pour le tri par insertion : au début du tour `i`, la partie `t[0:i]` est triée. C'est vrai au départ (un seul élément), chaque tour le conserve, et à la fin la partie triée est tout le tableau. Cette propriété qui reste vraie s'appelle un **invariant de boucle**.

Combien ça coûte

Les deux tris font, dans le pire des cas, environ $\frac{n^2}{2}$ comparaisons : c'est un **coût quadratique**. Doubler la taille du tableau multiplie le temps par 4. Pour 1 000 valeurs : 500 000 comparaisons ; pour un million : 5×10^{11}, beaucoup trop. Python utilise `sorted`, un tri en $n \log n$ bien plus rapide.

Le tri par insertion a un avantage : sur un tableau **presque trié**, il est très rapide.

Exemple guidé

Trie `[7, 3, 9, 1]` par sélection, en comptant les comparaisons.

Partie non triée	Minimum	Après échange
7, 3, 9, 1	1	1, 3, 9, 7
3, 9, 7	3	1, 3, 9, 7
9, 7	7	1, 3, 7, 9

1. À chaque étape, on cherche le minimum de la partie non triée et on l'échange avec sa première case.
2. On fait 3 comparaisons, puis 2, puis 1 : $3 + 2 + 1 = 6$, soit $\frac{n(n-1)}{2}$ pour $n = 4$.
3. Le tri par insertion sur le même tableau prend 3, puis 3 et 1 dans sa main de cartes : il fait moins de comparaisons quand le tableau est déjà presque trié.

Retour au problème

Pour un million de valeurs, $\frac{n(n-1)}{2}$ dépasse 5×10^{11} comparaisons. Les tris par sélection et par insertion sont quadratiques : parfaits pour 30 copies, trop lents pour un million. C'est pour ça que `sort` en Python utilise un algorithme plus rapide.

Les pièges

- Ces fonctions modifient le tableau reçu et ne renvoient rien.
- Dans le tri par sélection, on échange une fois par tour, après avoir trouvé le minimum.
- Coût quadratique : 10 fois plus de données, 100 fois plus de temps.

© Astrolab Cours · astrolabcours.fr/nsi/premiere/algorithmes-de-tri/ · Tous droits réservés : usage personnel uniquement, republication et usage commercial interdits