Exemples d'algorithmes de tri. Complexité.

903

Motivations : Trier est fondamental en informatique. Beaucoup d'algorithmes utilisent le tri comme étape. Les algorithmes de tri présentent aussi un intérêt théorique car ils mettent en œuvre des techniques classiques comme diviser pour régner.

Problème : On se donne un tableau T, de longueur n, d'éléments munis d'un ordre. On cherche une permutation des éléments par ordre croissant.

I. Un tri naïf : le tri par insertion

C'est le tri naturel utilisé pour trier un jeu de cartes.
À l'instant j, on insère l'élément $T[j]$ dans le sous-tableau $T[1..j-1]$ qui a déjà été trié.

```
TRI-INSERTION(T)
    Si T.longueur = 1 ne rien faire
    Sinon
        Pour j=2 à T.longueur
            clé = T[j], i = j-1
            Tant que i>0 et T[i]>clé
                T[i+1] = T[i], i = i-1
            T[i+1] = clé
```

Terminaison : chaque boucle 'Tant que' se termine car i décroît strictement.

Correction : On montre l'invariant de boucle : "Au début de chaque itération de la boucle 'Pour', le sous-tableau $T[1..j-1]$ contient les éléments initialement aux places $T[1]..T[j-1]$ mais qui sont triés".

Complexité : on évalue le nombre de comparaisons en fonction de la taille n du tableau T.

* pire des cas : tableau dans l'ordre décroissant : $C(n^2)$.
* en moyenne : on suppose qu'on compare à chaque étape $T[j]$ à la moitié des éléments du tableau $T[1..j-1]$ $C(n^2)$
* meilleur des cas : tableau déjà trié : $C(n)$.

Remarque : Ce tri est en place, il ne requiert pas d'espace mémoire supplémentaire que celui pour stocker T.

II. Tris en $O(n \log n)$

1) Tri fusion

Utilise la méthode "Diviser pour régner" : on trie séparément $T[1..\lfloor \frac{n}{2} \rfloor]$ et $T[\lfloor \frac{n}{2} \rfloor + 1..n]$ et on les fusionne.

```
TRI-FUSION(T,p,r) // trie T[p..r]
    Si p=r ne rien faire
    Si p<r
        q = ⌊(p+r)/2⌋
        TRI-FUSION(T,p,q)
        TRI-FUSION(T,q+1,r)
        FUSION(T,p,q,r)
```

```
FUSION(T,p,q,r)
    Copier T[p..q] dans L
    Copier T[q+1..r] dans R
    Ajouter ∞ à la fin de L et R
    i=1, j=1
    Pour k=p à r
        Si L[i] ≤ R[j]
            T[k] = L[i], i = i+1
        Sinon
            T[k] = R[j], j = j+1
```

Terminaison : cas d'arrêt dans la récursivité lorsqu'il n'y a qu'un élément

Correction : On montre celle de FUSION à l'aide de l'invariant : "Au début de chaque itération de la boucle 'Pour', le sous-tableau $T[p..k-1]$ contient les $k-p$ plus petits éléments de $L[1..q-p+2]$ et $R[1..r-q+1]$ en ordre trié, et $L[i]$ et $R[j]$ sont les plus petits éléments de L et R à ne pas avoir été copiés dans T".

Complexité : Avec la relation $C(n) = 2C(\lfloor \frac{n}{2} \rfloor) + O(n)$ on montre que $C(n) = O(n \log n)$ (en pire cas et en moyenne).

Inconvénient : Ce tri n'est pas en place (on copie les éléments du tableau dans FUSION).

2) Tri rapide

Repose également sur le principe "Diviser pour régner".
On commence par choisir un élément, appelé pivot, et on réorganise le tableau en plaçant à gauche du pivot tous les éléments inférieurs à lui et à droite les éléments supérieurs. On trie ensuite les deux sous-tableaux.

```
TRI-RAPIDE (T, p, r)   // Trie T[p..r]
| Si p=r ne rien faire
| Si p<r
|   q = PARTITION (T, p, r)
|   TRI-RAPIDE (T, p, q-1)
|   TRI-RAPIDE (T, q+1, r)
```

```
PARTITION (T, p, r)
| x = T[r], i = p-1
| Pour j = p à r-1
|   Si A[j] ≤ x
|     i = i+1, T[i] ↔ T[j]
| T[i+1] ↔ T[r]
| Retourner i+1 (indice du pivot)
```

Terminaison : même raison que le tri fusion.

Correction : On montre celle de PARTITION à l'aide de l'invariant :
"Au début de chaque itération de la boucle 'Pour', pour tout k :
1) Si $p \le k \le i$ alors $T[k] \le x$.
2) Si $i+1 \le k \le j-1$ alors $T[k] > x$.
3) Si $k = r$, alors $T[k] = x$"

Complexité : pire cas : tableau déjà trié. $O(n^2)$
en moyenne : $O(n \log n)$
Tri en place

3) Tri par tas (DVPMT)

Déf : Un tas est un arbre binaire dont tous les niveaux sont remplis sauf éventuellement le dernier où les éléments sont le plus à gauche possible. Un tas max est un tas où pour chaque sous-arbre, la racine est plus grande que ses descendants.

Le tri par tas s'effectue en 2 étapes : construire le tas (complexité linéaire), puis l'utiliser pour trier (complexité $O(n \log n)$).

Ce tri est en place.

III. Tris linéaires

1) Minorant pour les tris par comparaison

Th : Tout algorithme de tri par comparaison requiert $\Omega(n \log n)$ comparaisons dans le cas le plus défavorable

Preuve : avec un arbre de décision

Application : Tout algorithme de calcul de l'enveloppe convexe de n points s'exécute en temps $\Omega(n \log n)$. L'algorithme de Graham est en $O(n \log n)$ et utilise le tri.

Dans le plan !

Les tris que nous allons voir maintenant supposent des prérequis sur les entrées et utilisent d'autres méthodes que les comparaisons : ils auront une meilleur complexité.

2) Tri par dénombrement

Hypothèse : Les éléments de T sont dans $\{0, 1, \dots, k\}$, k fixé.

Principe : Compter pour chaque $i \in \{0, 1, \dots, k\}$ le nombre d'éléments de T inférieurs à i.

```
TRI-DENOMBREMENT (T, S, k)   // S contiendra le tableau trié
    Soit R[0..k] un nouveau tableau
    Pour i=0 à k
        R[i]=0
    Pour j=1 à T.longueur        // R[i] contient le nombre
        R[T[j]] = R[T[j]]+1      //   d'éléments égaux à i
    Pour i=0 à k                 // R[i] contient le nombre
        R[i] = R[i]+R[i-1]       //   d'éléments ≤ i, donc la place
                                 //   de i dans le tableau trié
    Pour j=T.longueur decr jusqu'à 1
        S[R[T[j]]] = T[j]        // S est une permutation triée
        R[T[j]] = R[T[j]]-1      //   de T
```

Complexité : $O(n+k)$.

Ce tri n'est pas en place mais est stable : deux éléments égaux apparaissent dans le même ordre en sortie qu'en entrée (peut être important si le tableau contient d'autres données)

3) Tri par base (tri radix)

Hypothèse : les entrées sont exprimées dans une base b avec au plus d chiffres.

Principe : On trie selon le chiffre de poids le plus faible, puis selon le suivant, en utilisant un tri stable (par exemple le tri par dénombrement puisque les chiffres sont dans $\{0..b-1\}$).

Complexité : avec un tri par dénombrement : $O(d(n+b))$.

Correction : par récurrence sur d

Rem : peut aussi servir pour trier une liste de mots de longueur bornée par ordre alphabétique.

4) Tri par paquets

Hypothèse : T contient des réels répartis uniformément dans $[0,1)$.

Principe : On divise $[0,1)$ en n sous-intervalles de même taille et on distribue les n nombres de T dans ces paquets. On trie ensuite les nombres de chaque paquet (en moyenne il y en a peu donc on utilise un tri par insertion)

```
TRI-PAQUETS (T)
    n = T.longueur
    Soit P[0..n-1] un nouveau tableau   // P contient les "paquets"
                                        //   sous forme de listes
    Pour i=0 à n-1
        P[i] = []
    Pour i=1 à n
        insérer T[i] dans la liste P[⌊n T[i]⌋]
    Pour i=0 à n-1
        trier P[i] avec un tri par insertion
    Concaténer les listes P[0], P[1], ..., P[n-1] dans l'ordre
```

Complexité : en moyenne : $\Theta(n)$

5) Recherche du i^e élément (DUPRAT)

On cherche seulement le i^e élément du tableau trié ; à l'aide d'une méthode 'Diviser pour régner' : On choisit astucieusement un pivot, on répartit les éléments autour de lui, et si le pivot n'est pas en place i on recommence dans le bon sous-tableau.

La complexité $T(n)$ vérifie : $T(n) \leq dn + T(\lfloor \frac{n}{5} \rfloor) + T(\lfloor \frac{7n}{10} \rfloor)$

donc on montre par récurrence que $T(n) = O(n)$.

Remarques: dommage d'écrire tous les algos - mais difficile de faire autrement
peut-être faire + de dessins, notamment pour les invariants de boucle.

Références : - Cormen
- J-D. Boissonnat, M. Yvinec : Géométrie algorithmique (borne inférieure pour l'enveloppe convexe)

Autres développements : Comparaison tri fusion / tri rapide, Complexité en moyenne du tri par paquets, borne inférieure de la recherche de l'enveloppe convexe + algorithme de Graham

Autres idées pour le plan : tri Shell, réseaux de tris

Q°: Quels algo utilisent le tri ?

-> Gram
-> algo géométrique (pt les + rapprochés)
-> arbre couvrant minimum (kruskall)

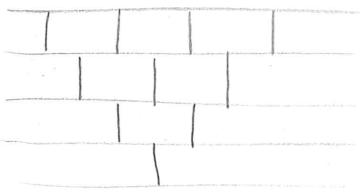
- Quoi utiliser pour tableaux de petit taille ? -> tri insert° (aussi bon quand presque trié)
- Ordre : ordre total -- ordre partiel ?
ici total c'est central.
si c'est partiel ~~-> ordre~~ tri topologique
(<-> ordre = graphe sans cycle.

---

tri shell (1959)
-> on fixe un pas h
-> on trie A[1, 1+h, 1+2h, ... 1+nh]
A[2, 2+h, ... , 2+nh]
A[h, 2h, 3h ... (n+1)h]
-> on recommence avec un h plus petit, jusqu'à h=1
complexité, résultats: cas moyen
pire cas: $O(n \log^2 n)$ pour $h_1, \dots h_n$ bien choisis

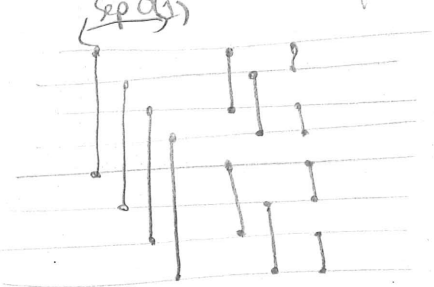
---

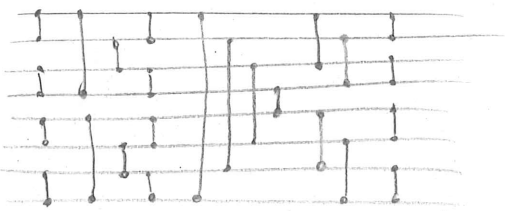
réseaux de tri (cf Cormen) algo parallèle.
Tri insert°
$O(n)$



---

tri bitonique





$C(1) = 0$
$C(n) = C(\frac{n}{2}) + \log n$
$= (\log n)\frac{(\log n + 1)}{2}$
$= O(\log^2 n)$

Il existe qqch en log mais la cst est tellement grande que c'est moins rapide en pratique.