Quelques methodes de tri parmi tant d'autres...

Généralités

Etant très fréquemment invoqué durant toutes sortes d'applications informatiques, le problème du *tri* d'ensembles (finis) d'éléments munis d'une relation d'orde total est crucial. Il existe de très nombreux algorithmes de tri et aucun d'entre eux ne peut être considéré comme « le meilleur ». Les conditions dans lesquels doit être appliqué le tri influeront sur le choix (souvent expérimental) d'un algorithme particulier.
Divers critères permettent de déterminer quelques grandes classes d'algorithmes de tri :

- On distingue le *tri interne*, où l'ensemble des éléments à trier peut être entièrement chargé dans la mémoire de l'ordinateur, et le *tri externe*, où l'ensemble des éléments à trier est trop grand pour pouvoir être traité en mémoire. Dans le cas du tri externe, on trie généralement des sous-ensembles tenant en mémoire, que l'on sauve dans des fichiers, puis on applique des méthodes de *fusion* de ces divers fichiers triés.
- Parmi les tris internes, on préfèrera souvent des tris *sur place*, qui n'utilisent qu'un (petit) nombre constant de variables auxiliaires, à des tris recopiant tout ou partie de la structure contenant les éléments à trier. Par exemple, on préfèrera généralement trier un tableau par des permutations au sein même du tableau plutot que de le recopier.
- Classiquement, on dégage 3 méthodes « naïves » principales de tri :
  - Les tris par *insertion* consistent intuitivement à prendre les éléments à trier les uns après les autres et à les insérer « à la bonne place » parmi les éléments déjà triés (un peu comme un joueur de cartes place ses cartes en main au fur et à mesure de la distribution).
  - Les tris par *sélection* consistent à dégager le plus grand (ou le plus petit) élément parmi ceux non encore triés, et à l'ajouter « au bout » de ceux déjà triés.
  - Les tris par *échanges* consistent à parcourir la liste des éléments à trier en effectuant un echange de places chaque fois que l'on rencontre 2 élements qui ne sont pas dans un bon ordre relatif. On arête les parcours lorsque plus aucun échange n'est à faire.

  Ajoutons les méthodes à la « divide and conquer », généralement très efficaces, qui partent du principe récursif suivant : trier un ensemble d'éléments, c'est le partitionner, trier chacune des parties puis en effectuer la fusion. En fin de récursion, des ensembles contenant 1 ou peu d'éléments ne sont pas durs à trier directement.

A titre d'exemple, voici un algorithme « naïf » de tri interne, sur place, par insertion. On remarquera néanmoins la technique de la *sentinelle* qui consiste à ajouter un élément fictif en bout de tableau qui assure l'échec de la condition (`x<a[j]`) du `while` avant que l'indice `j` ne dépasse les bornes du tableau. Ceci diminue considérablement le nombre de tests booléens car on écrit « `while x<a[j] ...` » au lieu de « `while x<a[j] and j>0 ...` ». (« `a` » est le tableau à trier)

```
var i, j: integer;
   x: valeur; (*n'importe quel type muni d'un ordre total*)

begin
   for i := 2 to n do begin
      x := a[i] ; a[0] := x ; j := i-1 ;
      while x < a[j] do begin
         a[j+1] := a[j] ; j := j-1
         end;
      a[j+1] := x
```

```
    end
  end.
```

On peut vérifier que le nombre de comparaisons est toujours compris entre $(n-1)$ et $(\frac{n(n+1)}{2}-1)$, et le nombre d'affectations entre $4(n-4)$ et $(4+n)(n-1)$ [n est le nombre d'éléments du tableau à trier].

Ce genre d'algorithme naïf utilise dans le cas le pire un temps de l'ordre de n^2 (nota : le nombre généralement grand d'éléments à trier justifie amplement de s'intéresser au comportement asymptotique).

Tris par tas

Le tri par tas est essentiellement un algorithme par sélection, mais il est l'un des (nombreux) algorithmes, plus rusés, en $O(n.log(n))$ dans le cas le pire. Ce qui lui confère cet avantage par rapport à un tri naïf par sélection est une gestion en *tournoi parfait* (=tas) de l'ensemble des éléments non encore sélectionés dans le cours de l'algorithme.

Pour cet exemple, on considère des tournois *croissants* le long des branches, si bien que la racine est le minimum. Le tri par tas comporte les étapes suivantes :

1) Constituer le tableau t des éléments à trier en un arbre tournoi parfait (en utilisant le rangement des arbres parfaits dans un tableau déjà décrit).

2) Supprimer le minimum du tournoi parfait (c'est la racine, i.e. l'élément indicé par 1 dans t), et le mettre à sa place dans le tableau en échangeant simplement $t[1]$ et $t[m]$ (où m est le nombre d'éléments non encore sélectionnés).

3) Réorganiser le tournoi parfait (car le nouvel élément placé à la racine durant l'étape **2)** n'est pas nécessairement le minimum du tournoi). Pour ce faire, il suffit d'échanger le nouvel élément avec le plus petit de ses fils, et itérer le processus (sur le fils en question) jusqu'à rétablir la propriété de tournoi.

4) Recommencer à l'étape **2)** jusqu'à ce que tous les éléments soient sélectionnés.

Tout ceci conduit à trier t par ordre décroissant ; choisir des tournois décroissant le long des branches permet bien sûr un tri croissant.

Pour effectuer l'étape **1)**, il suffit, pour chaque élément $t[i]$ du tableau $(t[1], t[2]\ \cdots)$ de l'échanger avec son père (c.a.d. $t[i/2]$) si nécessaire, et itérer le processus (avec le père) jusqu'à rétablir la propriété de tournoi pour les i premiers éléments de t.
On peut montrer que l'étape **1)** est en $O(n.log(n))$ telle qu'elle est décrite ici [en fait, il existe un algorithme linéaire pour cette étape]. L'étape **2)** ne demande aucune comparaison. L'étape **3)** est bien sûr en $O(log(m))$ chaque fois qu'elle est appelée. Tout ceci conduit à $O(n.log(n))$ après itérations de **2-4)**.