2.1 Introduction

2.1.1 Le tri

Trier des données est utile en informatique dans plusieurs cas : pour présenter des résultats (c'est alors l'opérateur humain qui tire parti du fait que les donnéee sont triées) ; pour optimiser des opérations (par exemple les recherches, dont la méthode dichotomique — celle que nous utilisons pour chercher dans un dictionnaire — repose crucialement sur le fait que les données sont triées) ; pour transformer les données (gérer ou supprimer les redondances, par exemple).

Ces exemples suffisent à faire comprendre l'importance du tri en algorithmique : manipuler de grands volumes de données est la fonction principale d'une grande partie de l'informatique. Mais il y a une autre raison pour laquelle on va s'intéresser assez longuement aux tris : les algorithmes les plus faciles à concevoir sont assez coûteux (d'ordre n^2), il y a donc un intérêt à tenter d'optimiser ces algorithmes, et par ailleurs, il existe de nombreux algorithmes, plus ou moins adaptés selon les cas, ce qui conduit à un travail fin de comparaison des algorithmes.

On peut vouloir trier toutes sortes de données, comme des mots dans un dictionnaire, et le plus souvent on veut trier des données complexes (fiches), ce qui nécessite d'introduire la notion de **clef**.

2.1.2 Notion de clef

Lorsqu'on trie des données informatiques, il s'agit généralement d'**enregistrements** qui comportent toutes sortes de données. Parmi ces données, il y a en général des informations permettant l'identification de l'enregistrement (nom, n° de code, côte...). C'est cela qu'on appellera la **clef**.

On fait souvent l'hypothèse que la clé est **unique**. Dans ce cas, si la clé apparaît plusieurs fois, on a plusieurs **occurrences** d'un enregistrement, et il s'agit de redondance. C'est une propriété explicitement recherchée pour les numéros de sécurité sociale, ou les cotes dans une bibliothèque, par exemple.

Mais on peut aussi accepter que la clé ne soit pas unique (c'est le cas des mots dans un dictionnaire, ou des noms d'une liste de clients). Dans ce cas, il peut être utile de faire intervenir une autre notion, celle de clé secondaire. On peut même imaginer d'utiliser des **clés structurées**. Dans ce cas, on peut distinguer les algorithmes de tri par une nouvelle propriété : la **stabilité** de l'algorithme. Un algorithme de tri est stable si l'ordre relatif des éléments de même clé n'est pas modifié.

Pour qu'on puisse parler de tri, il faut une relation d'ordre sur les clefs (i.e. une relation (de comparaison) antisymétrique et transitive).

Rq : Lorsque la réorganisation physique des enregistrements est coûteuse (gros enregistrements, tris sur mémoire de masse...), on peut trouver préférable de séparer les clés des enregistrements, formant ainsi ce qu'on appelle un *index*, sur lequel on peut faire le tri, à moindre coût. On parle de tri externe dans ce cas.

2.1.3 Structure de données linéaire

Dans tous les cas, l'objectif d'un tri est la réorganisation d'enregistrements de telle sorte que si on considère ces données dans l'ordre naturel (parcours), la valeur de leur champ clef forme une suite monotone croissante (ou décroissante).

Pourquoi parle-t-on d'ordre naturel ? parce que l'on travaille sur des structures de données linéaires. Il peut s'agir de tableaux (l'une des structures de données les plus simples en informatique, caractérisées par le fait que les enregistrements sont dans des cases mémoires contiguës), des listes (à la python), des fichiers, etc. Le point est qu'on peut indexer ces structures : on connaît le i^e enregistrement.

Dorénavant, on négligera très souvent le fait qu'on ne trie pas seulement les clés.

2.1.4 Evaluation et complexité

Pour choisir un algorithme de tri, on peut prendre en considération, comme pour la plupart des algorithmes, la complexité des différents algorithmes, qu'on peut évaluer en moyenne ou dans le pire des cas, en considérant le nombre d'opérations nécessaires, et la place nécessaire (les deux aspects habituels de la notion de complexité).

On ajoute souvent à ces critère des éléments d'évaluation spécifiques aux tris : d'une part des éléments de complexité particulièrement pertinents dans le cas des tris : le nombre de comparaisons (par définition dans un tri on compare souvent les clefs, et selon leur nature cela peut être plus ou moins coûteux) et le nombre de déplacements (quand on trie des enregistrements — et qu'on ne fait pas de tri externe — on peut être amené à déplacer souvent les enregistrements, ce qui peut représenter un coût en soi) ; d'autre part des propriétés qui rendent l'algorithme plus ou moins facile à utiliser. On peut citer la progressivité (est-ce qu'au cours du déroulement de l'algorithme il y a des parties qui sont déjà dans leur état définitif, ou le tableau risque-t-il au contraire d'être réorganisé à tout moment ?), la stabilité (est-ce que l'algorithme aboutit à des déplacements inutiles des enregsitrements ?) ou encore la sensibilité aux conditions initiales, en particulier au cas où les données sont presque dans l'ordre.

Ci-dessous une liste de ces critères permettant l'évaluation et la comparaison d'algorithmes de tri :

- nombre de cases mémoire (coût en place)
- nombre d'opérations élémentaires pour trier n enregistrements
  c'est la notion habituelle de complexité.
- nombre de comparaisons entre des clefs
  (les comparaisons peuvent être coûteuses : par exemple les chaînes de caractères)
- nombre de déplacements d'un enregistrement.
- stabilité (non dérangement des clés identiques = caractère non aléatoire)
- progressivité : tris où la partie triée est contiguë (*i.e.* définitive) et croissante, ce qui permet par exemple de commencer un traitement en parallèle sur le début.