HEC 922, 2001/2002 **TP 8 et 9**

1ère séance : 4/12, 11/12 et 12/12
2ème séance : 18/12, 19/12 * 2

Tris divers

Laure Danthony
`http://www.ens-lyon.fr/~ldanthon/`

Généralités, Objectifs

Le but de ce TP est d'étudier les algorithmes classiques de tri, que vous devez savoir programmer quasiment les yeux fermés. Bien sûr, on ne demande pas d'apprendre par cœur les codes, mais de connaître l'idée de chaque algorithme, et de retrouver rapidement comment le coder en Pascal.

On s'attachera également à analyser précisément les caractéristiques en terme de coût (nombre de comparaisons et nombre d'affectations, au pire, au mieux et en moyenne) des différents algorithmes.

On fournit dans le fichier `tri_squelette.pas` disponible sur le réseau des procédures de saisie et de visualisation de tableaux, qui ont été pompées du TP numéro 2.

1 Préliminaire

Dans les tris, on échange constamment deux cases d'un tableau.

▶ Ecrire une procedure `swap(i,j:integer;var t:table)` qui réalise l'échange des cases i et j du tableau t. Dans la suite, on désire trier le tableau $T = [T[1], T[2], \ldots T[n]]$ dans l'ordre croissant. Les éléments du tableau T ne seront pas nécessairement distincts.

2 Tri-bulle

On veut faire en sorte qu'après un premier passage, la dernière case du tableau contienne le maximum. Pour ce faire, on fait "remonter ce maximum" à partir de la première case du tableau jusqu'à la $(n-1)$-ème case, en échangeant de gauche à droite deux cases adjacentes si elles ne sont pas dans le bon ordre.

Au deuxième tour de boucle, on veut que le deuxième élément plus grand soit rangé dans la $n-1$-ième case. Pour ce faire, on réalise la même opération que précédemment, c'est à dire que l'on fait remonter à partir de $T[1]$ le deuxième élément plus grand en échangeant éventuellement deux cases adjacentes. Cette fois, on s'arrête ce processus à la $(n-2)$-ième case. Et on continue ...

Au j-ième tour de boucle on veut placer le j-ième plus rang élément à la $n-j+1$ case. Pour cela on le fait remonter de la gauche vers la droite en arrêtant les échanges à la $(j-1)$ème case.

▶ Coder la procédure `tri_bulle(var t:table)` qui réalise le tri-bulle sur le tableau t. Tester cette procédure sur des exemples pertinents. Evaluer sa complexité.

3 Tri-insertion

On suppose que le sous-tableau $T[1], \ldots T[k-1]$ est trié avec le tri insertion. On désire *insérer* placer l'élément $T[k]$. De 2 choses l'une :

- ou bien $T[k]$ est plus grand que tous les éléments $T[1], \ldots T[k-1]$, auquel cas il reste à sa place,
- ou bien il doit être inséré dans le sous tableau $T[1] \ldots T[k-1]$.

Une solution consiste à décaler l'élément $T[k]$ vers la gauche tant que possible en effectuant des échanges entre voisins.

▶ Coder la procédure `tri_insertion(var t:table)` qui réalise le tri par insertion du tableau t. Tester cette procédure. Evaluer la complexité.

4 Tri-sélection

Comme pour le tri-bulle, on désire qu'après un tour, le maximum du tableau soit à sa place. Cette fois par contre, on ne fait qu'un seul échange par boucle, la première fois on échange la case contenant le maximum et la case n du tableau. Au deuxième tour, on cherche le deuxième élément plus grand (dans les $n-1$ premières cases du tableau), et on l'échange à la fin avec la case $n-1$. Et ainsi de suite...

▶ Coder la procédure `tri_selection(var t:table)` qui réalise le tri par sélection du tableau t. Tester cette procédure. Evaluer la complexité.

5 Tri neuneu

On désire faire le tri récursif suivant : on trie les 2 premiers tiers, puis les 2 derniers tiers du tableau, et ainsi de suite.

▶ *En faisant bien attention aux indices mis en jeu et aux effets de bords*, coder la procédure `tri_neuneu(debut,fin:integer;var t:table)` qui réalise le tri-neuneu sur la tableau t entre les cases `debut` et `fin`. Vérifier qu'elle trie bien et justifier son nom.

6 Tri-fusion

On trie récursivement les 2 moitiés du tableau t, puis on fusionne comme il faut les deux sous-moitiés (étape difficile expliquée en cours de TD).

▶ Coder la procédure `tri_fusion(debut,fin : integer ; var t:table)` qui réalise le tri-fusion du tableau t entre les indices `debut` et `fin`. *Là aussi, on fera*

très attention aux problèmes d'indices. Faire des dessins ! Tester cette procédure sur des exemples pertinents. Evaluer la complexité.

7 Une petite amusette qui n'a rien à voir

On va se frotter ici à un "vrai" problème d'informaticien [1]. Dans la suite, on appelle **système monétaire** un tableau de m entiers vérifiant la relation $T[1] < T[2] < \ldots T[m]$ (ces entiers sont les valeurs faciales des "pièces"). On suppose que l'on dispose d'un nombre infini de pièces de chaque sorte.

▶ Exo : Donner un algorithme pour rendre la monnaie pour le système "Euro" : $[1, 2, 5, 10, 20, 50, 100, 200]$.

On nomme cet algorithme l'algorithme "glouton".

▶ Exo :
- Montrer que l'algorithme glouton n'est pas toujours optimal, *i.e.* trouver un système monétaire où il se trompe.
- (Si il reste du temps et que cela vous amuse) Coder l'algorithme.

Bonnes fêtes de fin d'année à tous !

[1] qui est encore source de nombreux articles