TP Info 5 | Révisions de programmation & algorithmes de tri

Cette séance de TP a pour objectif principal de réviser la structure de liste, les boucles, la terminaison, les invariants de boucle, la terminaison et la notion de complexité au travers de quelques algorithmes classiques de tri (qui figurent au programme de seconde année).

1. Ce qu'il faut savoir

Révision de tous les épisodes précédents.

2. Exercices

Les difficultés sont échelonnées de la manière suivante : aucune, ♪ , ♪♪ , ♪♪♪ et ♪♪♪♪. Certains énoncés sont tirés des annales des concours (oral et écrit) ; leur provenance est le plus souvent précisée. Les exercices notés ♪♪♪ et ♪♪♪♪ sont particulièrement délicats.

2.1. Algorithmes de tri

1 . [*Tri par sélection* ♪]
Notons n la longueur de la liste `t`. Le tri par sélection s'effectue de la manière suivante : on commence par déterminer le plus petit parmi `t[0]`, ..., `t[n-1]` puis on échange `t[0]` et cet élément. On continue par déterminer le plus petit élément parmi `t[1]`, ...,`t[n-1]` puis on échange `t[1]` et cet élément, et on continue ainsi de suite jusqu'à épuisement de la liste.

a) Mettre en œuvre cet algorithme *à la main* sur `t=[6,-3,7,5,1,0]`.

b) Écrire une procédure `triSelection(t)` renvoyant la liste `t` triée selon cette méthode.

c) Prouver la terminaison et la correction de cette fonction.

d) Déterminer la complexité de cet algorithme.

2 . [*Tri par insertion* ♪]
Le tri par insertion est un algorithme de tri que la plupart des personnes utilisent naturellement pour trier des cartes : prendre les cartes mélangées une à une sur la table, et former une main en insérant chaque carte à sa place. Le tableau est trié en place, ie on ne crée pas de tableau auxiliaire, on *travaille* directement sur le tableau initial.

a) Mettre en œuvre cet algorithme *à la main* sur `t=[6,-3,7,5,1,0]`.

b) Écrire une fonction `triInsertion(t)` renvoyant la liste `t` triée selon cet algorithme.

c) Prouver la terminaison et la correction de cet algorithme.

d) Calculer la complexité de cette fonction dans le meilleur des cas puis dans le pire des cas.

3 . [*Tri-bulle* ♪]
Le principe du tri à bulle est le suivant : parcourir le tableau en comparant les couples d'éléments successifs. Lorsque deux éléments successifs ne sont pas dans l'ordre croissant, ils sont échangés : le maximum arrive en position finale en remontant comme une bulle de champagne. On recommence alors l'opération jusqu'à obtenir une liste entièrement triée.

a) Mettre en œuvre cet algorithme *à la main* sur `t=[6,-3,7,5,1,0]`.

b) Écrire une fonction `triBulle(t)` renvoyant la liste `t` triée par cette méthode.

c) Prouver la terminaison et la correction de cet algorithme.

d) Quelle est la complexité dans le meilleur des cas puis dans le pire des cas de cet algorithme ?

4 . [*Tri par dénombrement* ♪]
Soit N un entier naturel. On suppose, dans cette partie, que les listes sont constituées d'entiers compris entre 0 et N. Écrire une fonction `triPostier(t)` qui trie une liste `t` en temps linéaire par rapport à sa longueur. On pourra utiliser une liste auxiliaire `tcomptage` qui compte les occurrences de chaque entier de $[\![0,\mathrm{N}]\!]$.

2.2. Compléments

5 . [*Recherche dichotomique dans une liste triée* ♪]
En suivant l'algorithme de recheche dichotomique dans une liste triée, écrire une procédure `rechercheDicho(t,a)` renvoyant `True` si l'élément a appartient à la liste triée `t` et `False` sinon.

6 . [*La suite de Conway* ♪♪]
Le premier terme de la suite de Conway est posé comme égal à 1. Chaque terme de la suite se construit en annonçant le terme précédent, c'est-à-dire en indiquant combien de fois chacun de ses chiffres se répète. On représente les termes de la suite par des listes. Ainsi, $u_0 =$ `[1]`. Ce terme comporte juste un « 1 ». Par conséquent, le terme suivant est $u_1 =$ `[1,1]`. Celui-ci est composé de deux « 1 », on a donc $u_2 =$ `[2,1]`, et, en poursuivant le procédé :

$$\begin{cases} u_3 = \texttt{[1,2,1,1]} \\ u_4 = \texttt{[1,1,1,2,2,1]} \\ u_5 = \texttt{[3,1,2,2,1,1]} \\ u_6 = \texttt{[1,3,1,1,2,2,2,1]} \end{cases}$$

et ainsi de suite.

a) Écrire une procédure `suivantConway(t)` renvoyant la liste obtenue en appliquant le procédé décrit ci-dessus à la liste `t`.

b) Afficher les dix premiers termes de la suite de Conway.

7 . [*Algorithme glouton du paiement* ♪♪]
Considérons le problème suivant : payer une somme en euros avec des pièces de 2, 1, 0.5, 0.2, 0.1, 0.05 et 0.01 euros. On étudie dans cet exercice l'algorithme glouton consistant à répéter le choix de la pièce de plus grande valeur qui ne dépasse pas la somme restante.

a) Écrire une fonction `glouton(x)` renvoyant une liste de couples `[valeur de la pièce, nombre de pièces]` correspondant à l'algorithme glouton pour la somme de `x` euros.

b) Cet algorithme renvoie-t-il toujours une solution optimale, c'est-à-dire comportant le moins possible de pièces ?

Plus généralement, un algorithme est dit *glouton* s'il suit le principe de faire, étape par étape, un choix optimum local, dans l'espoir d'obtenir un résultat optimum global. Un algorithme n'est pas nécessairement optimal.