Programmation en Fortran et C++

1 Un algorithme de tri : *heapsort*

Dans bien des cas, nous avons besoin de réarranger les éléments d'un tableau d'objets en ordre croissant (ou décroissant). L'algorithme de tri par tas, ou *heapsort* en anglais, est très populaire à cause de son efficacité : le nombre d'opérations nécessaires pour trier un tableau A de taille n est toujours proportionnel à $nlog_2 n$. Sans vouloir développer la théorie des algorithmes de tri (qui fait l'objet des manuels d'informatique), nous considérons utile de donner au lecteur une idée sur le fonctionnement de cet algorithme. La notion de base est celle d'arbre ordonné (tas ou *heap*), il s'agit d'un arbre binaire (voir figure 1) pour lequel la valeur en chaque noeud N est inférieure ou égale à celle de tout descendant de N, autrement dit

$$A(j/2) \leq A(j), \quad \forall\, 1 \leq j/2 < j \leq n, \text{ où } j/2 \text{ est calculé par une division entière.}$$

Une fois que le tas est formé (figure 1) un algorithme de suppression de l'arbre est appliqué pour

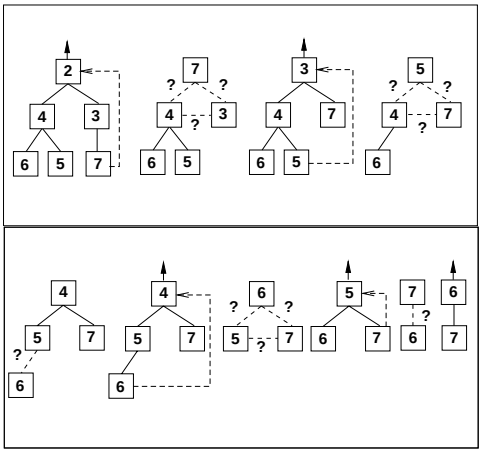

FIGURE 1 – *Heapsort* : phase de suppression.

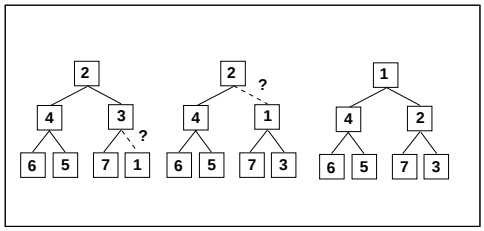

FIGURE 2 – *Heapsort* : phase de construction du tas.

ordonner les éléments du tableau :
- on commence par supprimer le noeud « racine » de l'arbre (qui est le minimum des toutes les valeurs stockées dans l'arbre) ;
- sa place sera prise par le dernier élément de l'arbre (remplissage vers la gauche) ;
- le nouveau noeud est comparé avec ses deux descendants directs et, éventuellement, remplacé par le plus petit ; cette opération est propagée le long de la descendance, chaque fois qu'il y a permutation entre deux noeuds ;
- on s'assure ainsi que la structure de tas est rétablie et on peut donc supprimer la nouvelle racine.

Mais, pour appliquer l'algorithme de suppression il faut savoir comment construire le tas. La figure 2 montre comment insérer un nouvel élément dans notre arbre ordonné. Il est rajouté à la fin de l'arbre et par comparaisons successives avec son ascendant direct, on fait « migrer » la nouvelle valeur à sa position appropriée, afin que l'arbre redevienne un tas.

Les deux algorithmes, d'insertion et de suppression, constituent la base de l'algorithme de tri de type *heapsort*.

2 Résolution de systèmes linéaires : algorithme du *gradient conjugué*

Pour résoudre le système linéaire $Ax = b$, avec $A \in \mathbb{R}^{n\times n}$ une matrice symétrique, définie positive, la méthode du gradient-conjugué (GC) est très efficace. Une formulation *optimisée* de l'algorithme est la suivante :

Algorithme 1

Le gradient conjugué pour résoudre le système linéaire $Ax = b$:
soient $x^0 \in \mathbb{R}^n$ la valeur initiale (donnée) et ε la précision (fixée)
$g^0 = Ax^0 - b$
$h^0 = -g^0$
– pour $i = 0$ à n

$\rho = -\dfrac{(g^i, h^i)}{(h^i, Ah^i)}$	*(pas optimal)*
$x^{i+1} = x^i + \rho h^i$	*(avancement suivant h^i)*
$g^{i+1} = g^i + \rho A h^i$	*(calcul itératif du gradient)*
$\gamma = \dfrac{(g^{i+1}, g^{i+1})}{(g^i, g^i)}$	*(paramètre assurant $(h^{i+1}, h^i) = 0$)*
$h^{i+1} = -g^{i+1} + \gamma h^i$	*(calcul itératif du la direction de descente)*
si $(g^{i+1}, g^{i+1}) < \varepsilon$ *stop*	

Si l'algorithme (GC) peut être considéré comme une méthode exacte (la convergence étant assurée en n itérations au plus), il est généralement utilisé comme méthode itérative en s'attendant à ce qu'il converge plus rapidement. Pour accélérer la vitesse de convergence, on peut *préconditionner* le système linéaire. L'idée ici est de multiplier le système initial par une matrice C, dite de *préconditionnement*, et résoudre le nouveau système ($CAx = Cb$) par le (GC) classique.

Exercice 1 (Algorithme de tri – programmation en Fortran)
Copier dans votre répertoire personnel le fichier
http://www.ann.jussieu.fr/∼danaila/zdownload/DEA_2011/dea_exemples_fortran/dheap.f
qui implémente l'algorithme de tri sous forme de `subroutine`.

1. Ecrire un programme Fortran et tester l'algorithme pour trier un tableau rempli avec des valeurs aléatoires.
2. Même question, mais avec lecture des éléments du tableau à partir d'un fichier (et écriture du résultat dans un fichier différent).
3. Reprenez la première question et effectuer le tri de chaque colonne d'une matrice carrée.

Exercice 2 (Algorithme de tri – programmation en C++)
Reprenez l'exercice précédent, mais en écrivant les programmes en C++. La fonction qui implémente l'algorithme de tri se trouve dans
http://www.ann.jussieu.fr/∼danaila/zdownload/DEA_2011/dea_exemples_cpp/heap.hpp
Utiliser la même fonction pour trier des objets plus abstraits en illustrant ainsi les possibilités de programmation orientée objet.

Exercice 3 (Algorithme du gradient conjugué)

1. Ecrire une `subroutine` Fortran pour implémenter l'algorithme du gradient conjugué. Tester le programme.
2. Même question pour la programmation en C++. Un exemple d'implémentation est donné dans http://www.ann.jussieu.fr/∼danaila/zdownload/DEA_2011/dea_exemples_cpp