Colle n° 3 d'Informatique – CaML

Mathieu Goutelle
Mathieu.Goutelle@ens-lyon.fr

18 – 22 novembre 2002

Nous allons étudier ici quelques tris classiques sur les listes : le tri par insertion, le tri par tas (*HeapSort*), le tri fusion (*MergeSort*) et le tri dit « rapide » (*QuickSort*). La partie « arbre » de cette colle concerne particulièrement le tri par tas.

1 Préliminaire

Écrire une fonction générant une liste aléatoire d'entiers. Elle vous servira pour générer des échantillons pour les tests. Le nom des fonctions à connaître du module `random` de CaML sont `init : int -> unit = <fun>` (initialisation du générateur de nombres pseudo-aléatoires) et `int : int -> int = <fun>` (génération d'un entier pseudo-aléatoire tel que `int k` génère un entier entre 0 et $k-1$).

2 Le tri par insertion

2.1 Principe
On suppose la liste triée et un nouvel élément arrive : il faut l'insérer à sa place dans la liste. Quelle est la complexité de ce tri ?

2.2 Implémentez le tri par insertion et tester sur quelques échantillons.

3 Le tri par tas (*HeapSort*)

Voilà la partie arbre de ce TD : on appelle *tas* un arbre semi-ordonné, tel que la racine de l'arbre soit inférieure ou égale aux nœuds de ces fils gauche et droit et que ces deux sous-arbres soient aussi des tas. Le type CaML utilisé sera défini ainsi :

```
type 'a Tas =
    |Rien
    |Noeud of 'a * 'a Tas * 'a Tas
;;
```

3.1 Écrire une fonction `racine : 'a Tas -> 'a = <fun>` qui renvoie la racine du tas.

3.2 Écrire une fonction `ajouter : 'a -> 'a Tas -> 'a Tas = <fun>` qui ajoute un élément à un tas déjà existant. Pour éviter de déséquilibrer l'arbre après de nombreuses insertions, le fils gauche du nouveau tas est le fils droit de l'ancien et inversement. Ainsi, on ajoute l'élément « toujours à droite », mais on permute à chaque insertion droite et gauche. On aurait bien sûr pu insérer toujours à gauche tout en permutant, ou encore choisir droite et gauche de façon aléatoire à chaque insertion mais sans permuter cette fois. Cela part du même principe dit de *randomisation*...

3.3 Écrire une fonction `enlever : 'a Tas -> 'a Tas = <fun>` qui enlève l'élément à la racine du tas. Pour cela, on supprime la racine du tas que l'on remplace par le minimum des nœuds des fils gauche et droit. On supprime alors récursivement la racine choisie du sous-arbre.

3.4 Écrire une fonction `list_to_heap : 'a list -> 'a Tas -> 'a Tas = <fun>` qui ajoute les éléments d'une liste à un tas existant, puis une fonction `heap_to_list : 'a Tas -> 'a list = <fun>` qui rend la liste triée des éléments contenus dans le tas et finalement une fonction `heapsort : 'a list -> 'a list = <fun>` qui applique les fonctions précédentes pour trier une liste.

4 Le tri fusion (*MergeSort*)

4.1 Principe

Pour trier une liste, on peut la séparer en sous-listes que l'on va trier puis fusionner. Quelle est la complexité de ce tri ?

4.2 Écrire une fonction de fusion de deux listes triées.

4.3 Implémentez le tri fusion et tester sur quelques échantillons. Il existe de multiples manières, plus ou moins efficaces, de programmer ce tri. On pourra par exemple séparer la liste en des listes à un élément, pour les fusionner ensuite. Une des meilleures solutions est celle proposée dans Aho-Ullmann (cf. explications au tableau et corrigé)...

5 Le tri rapide (*QuickSort*)

5.1 Principe

On choisit un élément de la liste que l'on désigne comme *pivot*. On sépare ensuite la liste selon les éléments inférieurs, égaux et supérieurs au pivot. Il reste alors à trier les deux listes produites. Quelle est la complexité de ce tri ?

5.2 Implémentez le tri fusion et tester sur quelques échantillons.

6 Comparaisons

Comparer les temps d'exécution des quatre tris précédents sur une liste longue (10 000 éléments par exemple). Voilà un exemple pour 10 000 éléments (Duron 800MHz, CaML Light 0.74) :

Type de tri	Durée (s)
Tri par insertion	36,393
Tri par tas	0,511
Tri fusion #1	0,35
Tri fusion #2	0,401
Tri rapide	0,4