IN 101 - TD 12
9 décembre 2011

Énoncé

ENSTA ParisTech

Exercice 1 : Implémentation d'un tas

Comme vu en cours, un tas est un arbre binaire complet, ordonné selon la règle suivante : un nœud possède une clef supérieure à toutes celles des nœud du sous-arbre dont il est la racine. La racine d'un tas est donc toujours le nœud ayant la clef la plus grande. Un tas permet donc de gérer des *files de priorité* : l'élément de priorité maximale sera alors toujours à la racine, prêt à être extrait du tas pour être traité.

1.a] En notant h la hauteur du tas, exprimez la complexité des opérations de recherche du maximum, d'insertion d'un élément et de suppression de l'élément maximum en fonction de h.

1.b] Exprimez h en fonction du nombre d'éléments n contenus dans le tas. En déduire la complexité des trois opérations en fonction de n.

Implémentation d'un tas. Les tas peuvent se coder comme des arbres, mais cela rend les opérations assez complexes. On préfère donc une implémentation avec un tableau qui sera plus efficace en pratique.

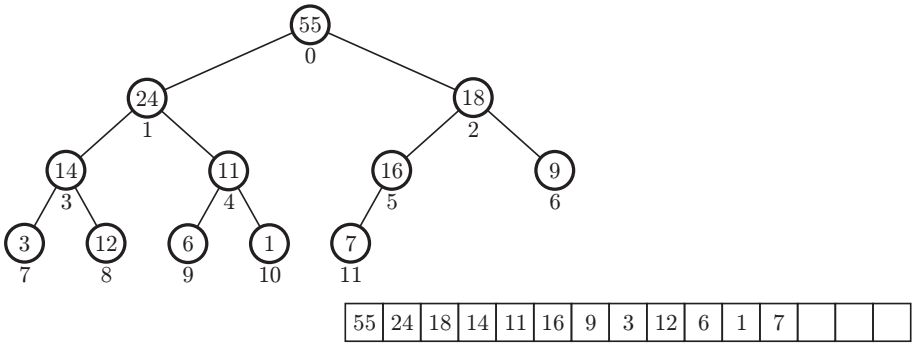

FIGURE 1 – Représentation d'un tas par un tableau.

Avec cette représentation, le calcul de chaque fils et du père d'un nœud est simple : le fils gauche du nœud d'indice i est, s'il existe, à l'indice $2i + 1$, le fils droit à l'indice $2i + 2$, et le père en $\lfloor \frac{i-1}{2} \rfloor$. En C, un tas est (un pointeur vers) une `struct` contenant un tableau `tab`, un entier `max` représentant le nombre maximum d'éléments et un entier `n` représentant le nombre d'éléments effectivement stockés :

```
1 typedef struct {
2   int max;
3   int n;
4   int* tab;
5 } heap;
```

1.c] Implémentez les tas au moyen de tels tableaux, et programmez les opérations associées de création, d'impression (ce n'est pas une opération standard, mais ça peut aider pour débugger), de recherche du maximum et d'insertion (on ne programme pas l'extraction pour le moment). Voici les signatures des fonctions que vous devez implémenter.

```
1 /* Création d'un tas vide de taille m (et allocation de la mémoire) */
2 heap* init_heap (int m) {...}
3 /* Impression (en largeur) du contenu d'un tas */
4 void print_heap (heap* H) {...}
5 /* maximum d'un tas */
6 int heap_maximum (heap* H) {...}
7 /* insertion de la clef k dans un tas */
8 void heap_insert (int k, heap* H) {...}
```

Programmez une fonction `main` qui initialise un tas pour contenir jusqu'à 31 éléments et y insère les éléments de l'arbre de la figure 1, dans l'ordre. Insérez y ensuite l'élément 43 et vérifiez que l'ordre des nœuds est toujours bon.

1.d] Ajoutez à cela la fonction de suppression (avec retour du maximum) ayant comme signature `int heap_delete (heap* H)`. Dans le `main`, supprimez la racine de votre tas, vérifiez que cela vous retourne bien 55 et qu'ensuite, les nœuds du tas sont bien dans un ordre qui convient.

Exercice 2 : Tri par tas

Un algorithme de tri efficace, appelé *tri par tas* (*heapsort* en anglais) peut être déduit de la construction précédente : il consiste à ajouter chaque élément du tableau à trier dans un tas, puis à retirer un à un les éléments du tas.

2.a] Quelle est la complexité du tri par tas ?

2.b] Programmez ce tri. Il devra modifier le tableau `tab` passé en argument pour en réordonner les éléments. La signature de la fonction est `void heapsort (int* tab, int n)`. Attention, ce tri doit être fait « en place, » en allouant une taille mémoire supplémentaire indépendante de n. Pour cela, le même tableau contient les éléments du tas et ceux non encore insérés dans le tas. Une fois le tas remplis, il faut en extraire les éléments et les placer à la fin du tableau `tab`.

Exercice 3 : Recherche des k plus grands éléments

Pour chercher le plus grand élément d'un tableau, il suffit de parcourir le tableau une fois. Si on remplace le tableau par un flux de nombre (que l'on ne peut lire qu'une fois, dont on ne connaît pas la taille à l'avance et qui peut avoir une taille très grande) l'algorithme ne change pas : on garde en permanence l'élément le plus grand que l'on a déjà rencontré. Maintenant, si on veut trouver les k plus grands éléments d'un tableau la meilleure méthode permet d'obtenir une complexité de $\Theta(n + k \log k)$ avec l'algorithme "quick select" suivi d'un tri des k plus grands éléments. En revanche, cette technique ne s'applique pas à un flux de données.

3.a] Expliquez comment utiliser un tas pour extraire les k plus grands éléments d'un flux de nombres avec une complexité en temps de $\Theta(n \log k)$ et en mémoire de $\Theta(k)$.

3.b] Implémentez cet algorithme pour extraire et afficher (dans l'ordre croissant) les 10 plus grands entiers du fichier `~finiasz/nombres`. Vérifiez qu'il vous affiche bien :

```
32447 32456 32557 32582 32594 32598 32618 32622 32731 32734
```

Exercice 4 : Implémentation d'un arbre AVL

Nous allons implémenter une structure d'arbre AVL (sauf la suppression). Pour cela, nous avons besoin d'une structure d'arbre binaire de recherche à laquelle nous ajoutons dans chaque nœud la hauteur du sous-arbre dont ce nœud est la racine (0 si il n'a pas de fils) et le coefficient d'équilibrage correspondant à la différence de hauteur entre le sous-arbre droit et le sous-arbre gauche de ce nœud (un sous-arbre vide a une hauteur de -1).

4.a] Créez une structure adaptée pour représenter un nœud d'un arbre AVL.

4.b] Implémentez une fonction *void* `insert(`*node*`** A, `*int*` v)` qui insère la valeur `v` dans l'arbre ayant pour racine `*A`. Pour l'instant cette fonction devra simplement insérer le nouveau nœud et mettre à jour les hauteurs d'arbres et les coefficients d'équilibrage.

4.c] Implémentez les fonctions de rotation *void* `rot_G(`*node*`** A)` qui effectue une rotation à gauche en partant du nœud `*A` et *void* `rot_D(`*node*`** A)` qui effectuent une rotation à droite. Attention, ces deux fonctions doivent aussi mettre à jour les hauteurs et les coefficients d'équilibrage des deux nœuds modifiés.

4.d] Modifiez votre fonction d'insertion pour qu'elle effectue une rotation à gauche, une rotation à droite ou une double rotation pour rétablir la propriété d'arbre AVL quand le coefficient d'équilibrage passe à 2 ou -2.