Option Informatique
TP3 : Représentations contigües d'un arbre, files de priorité, tas

Sujet

1 Représentation contigüe d'un arbre

On considère un arbre binaire au sens large T dont les sommets sont des entiers. On se propose d'étudier plusieurs implémentations d'un tel arbre et des opérations élémentaires sur les arbres. Dans la suite, on considère l'arbre T de la figure 1.

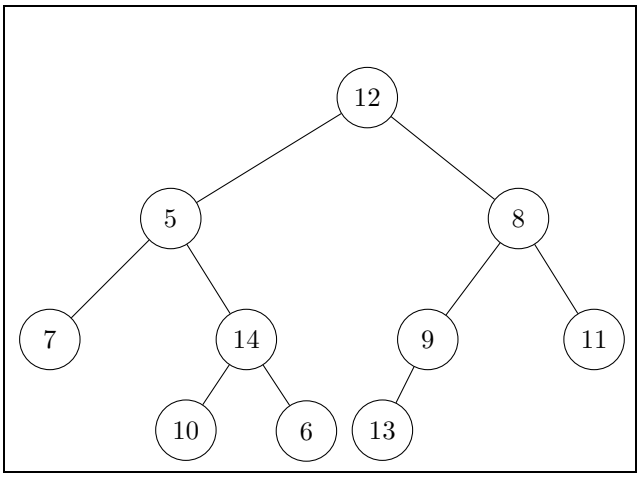

FIG. 1 – L'arbre T.

1.1 Utilisation d'un tableau

Chaque sommet de l'arbre est représenté par un enregistrement contenant trois champs : un champ *gauche* indiquant l'indice du fils gauche dans un tableau, un champ *droit* indiquant l'indice du fils droit et un champ *info* indiquant la valeur associée au sommet.

La valeur `nil` est associée à -1. On définit ainsi le type :

```
type element = { droit : int ; info : int ; gauche : int} ; ;
```

L'arbre T est représenté à l'aide d'un tableau et d'une variable racine indiquant l'indice de la racine dans le tableau. On définit le type :

```
type arbre = { racine : int ; taille : int ; sommet : element vect} ; ;
```
Ainsi l'arbre précédent peut être représenté de la façon suivante :
```
let T = {racine = 0 ; taille = 10 ; sommet=[|{droit = 3 ; info = 12 ; gauche = 2} ;...
 ; {droit = -1 ; info = 13 ; gauche = -1}|]}
```
où `T.sommet` est le tableau représenté en figure 2.

gauche	2	4	6	-1	8	10	-1	-1	-1	-1
info	12	5	8	7	14	9	11	10	6	13
droite	3	5	7	-1	9	-1	-1	-1	-1	-1

FIG. 2 – Une représentation de `T.sommet`

Écrire les opérations élémentaires `est_terminal`, `racine`, `fils_gauche`, `fils_droit`, `cons : arbre → int → arbre → arbre` *avec cette implémentation d'un arbre.*

1.2 Une structure plus simple

On va maintenant définir le tableau représentant notre arbre de la manière suivante :

```
type tas = mutable taille : int, contenu : int vect ;
let nouveau_tas n =  nombre = 0 ; contenu = make_vect n 0 ;
```

Cette numérotation va permettre de stocker les éléments de l'arbre dans un tableau de longueur n. On peut alors remarquer que les fils de l'élément noté i dans le tableau sont l'élément numéroté $2i+1$ pour le fils gauche et $2i+2$ pour le fils droit (en notant que les indices du tableau varient de 0 à $n-1$). Comme le nombre d'éléments de notre tableau sera amené à varier par la suite, nous préférons utiliser un tableau de grande taille dont nous n'utiliserons que les `t.taille` premiers éléments.

Écrire les fonctions `arbre_en_tas` *et* `tas_en_arbre` *qui transformetn respectivement un arbre en tas et un tas en arbre.*

On donne le type arbre : `type arbre = Vide | Noeud of arbre * int * arbre ;`

2 Files de priorité et tas

On suppose, que des gens se présentent au guichet d'une banque avec un numéro écrit sur un bout de papier représentant leur degré de priorité. Plus ce nombre est élevé, plus ils sont importants et doivent passer rapidement. Bien sûr, il n'y a qu'un seul guichet ouvert, et l'employé(e) de la banque doit traiter rapidement tous ses clients pour que tout le monde garde le sourire. La file des personnes en attente s'appelle une file de priorité. L'employé de banque doit donc savoir faire rapidement les 3 opérations suivantes : trouver un maximum dans la file de priorité, retirer cet élément de la file, savoir ajouter un nouvel élément à la file. Plusieurs solutions sont envisageables.

Une méthode élégante consiste à gérer une structure d'ordre partiel grâce à un arbre. La file de n éléments est représentée par un arbre binaire contenant en chaque noeud un élément de la file. L'arbre vérifie deux propriétés importantes : d'une part la valeur de chaque noeud est supérieure ou égale à celle de ses fils, d'autre part l'arbre est quasi complet.

un arbre binaire de hauteur h est quasi-complet si pour tout $d \in [0, h-1]$, le niveau d de l'arbre possède 2^d éléments et si les noeuds du niveau h sont les plus à gauche possible.

Un tas est un arbre quasi-complet qui est une file de priorité.

2.1 Insertion dans une file de priorité

Écrire la fonction d'échange de deux cases dans un tableau

L'ajout d'un nouvel élément v à la file consiste à incrémenter `taille`, puis à poser `t.contenu[n]=v`.

Ceci ne représente plus un tas car la relation n'est pas nécessairement satisfaite. Pour obtenir un tas, il faut échanger la valeur contenue au noeud n et celle contenue par son père, remonter au père et réitérer jusqu'à ce que la condition des tas soit vérifiée. Ceci se programme par une simple itération.

Écrire la procédure ***`insere x h`*** *qui ajoute un élément* ***`x`*** *à une file de priorité* ***`h`****.*
En supposant que le tas q n éléments, quelle est la complexité de cette fonction ?

2.2 Supression dans une file de priorité

Considérons l'opération de suppression du premier élément de la file. Il faut alors retirer la racine de l'arbre représentant la file, ce qui donne deux arbres ! Le plus simple pour reformer un seul arbre est d'appliquer l'algorithme suivant : on met l'élément le plus à droite de la dernière ligne à la place de la racine, on compare sa valeur avec celle de ses fils, on échange cette valeur avec celle du vainqueur de ce tournoi, et on réitère cette opération jusqu'à ce que la condition des tas soit vérifiée. Bien sûr, il faut faire attention, quand un noeud n'a qu'un fils, et ne faire alors qu'un petit tournoi à deux.

On définit ainsi une opération inverse de la percolation vers le haut définie ci-dessus.

La fonction $\texttt{supprime}_{racine}$ s'écrit donc :

```
let supprime_racine h = let n = h.nombre and racine = h.contenu.(0) in
  h.contenu(0) <- h.contenu(n+1) ;
  h.nombre <- (n-1) ;
  percole h ;
where percole h = ...
```

Écrire la fonction ***`percole`*** *qui consiste à faire redescendre l'étiquette de la racine tout au long du tas en échangeant récursivement avec la plus grande des étiquettes de ses fils, jusqu'à ce que son étiquette soit plus grande que toutes les étiquettes de ses fils.*

2.3 Recherche du maximum dans une file de priorité

Comment obtient-on le maximum de la file de priorité ?

2.4 Tri par tas

Écrire la fonction ***`tri_en_tas`*** *qui donne la séquence triée des éléments contenus dans un tas* `t`*.*

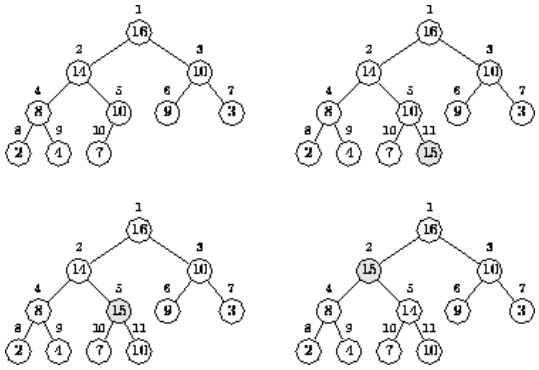

FIG. 3 – ajout dans une file de priorité

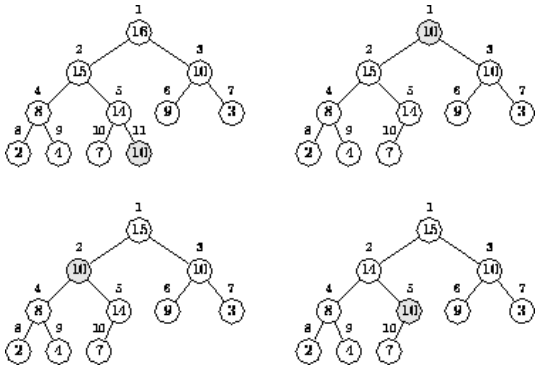

FIG. 4 – Supression dans une file de priorité