Leçon 4 : Tris par tas

AYIKPA KACOUTCHY JEAN : Enseignant - Chercheur

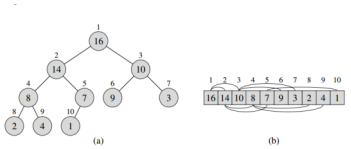
Table des matières

1- Généralité sur le tri par tas

I

La structure de tas (binaire) est un tableau qui peut être vu comme *un arbre binaire presque complet*.

Chaque *nœud de l'arbre* correspond à un élément du *tableau* qui contient *la valeur du nœud*.



La valeur de chaque nœud est à l'intérieur du cercle à ce nœud.

Le nombre au dessus de chaque nœud est l'indice correspondant dans le tableau A.

Le tableau A représente un tas à deux attributs :

- *longueur(A)* : nombre d'éléments.
- *taille(A)* : nombre d'éléments du tas rangé dans le tableau

Si i est l'indice d'un nœud, les indices de son père, fils gauche et fils droit se calculent comme suit :

- *Père*
  Fonction pere({E} i : entier) : entier
  début
  renvoyer (i mod 2)
  fin
- *Fils gauche*
  Fonction gauche({E} i : entier) : entier
  début
  renvoyer (2*i)
  fin
- *Fils droit*
  Fonction droit({E} i : entier) : entier
  début
  renvoyer (2*i+1)
  fin

Les tas satisfont également la propriété de tas : pour chaque nœud i autre que la racine,

- *A [pere(i)] >= A[i]*

On peut définir deux sortes de tas binaires : les tas min et les tas max.

- *Tas-min* : chaque élément est supérieur à son parent, le plus petit élément d'un tas min est à la racine.
- *Tas-max* : chaque élément est inférieur à son parent, ainsi, le plus grand élément d'un tas max es stocké dans la racine.

NB : Pour l'algorithme du tri par tas, on utilise des tas max. Les tas min servent généralement dans les files de priorités.

Application 1 :

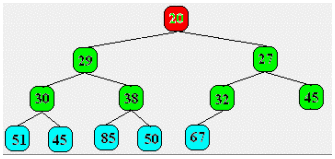
II

Exercice

La structure de tas est :

- un sous arbres
- un tableau qui peut être vu comme un arbre binaire
- un sous-arbres d'un arbre binaire

Exercice



- A partir de la fonction du fils gauche veiller donner la valeur de de son indice et la valeur du nœud pour i =5

  indice = valeur du nœud =

- Veiller remplir le tableau de tas correspondant à l'arbre ci-dessus :

2- Construire un tas et Algorithme de tri par tas

2.1- Conservation de la structure de tas

ENTASSER est un sous-programme permettant la manipulation des tas.

Il prend en entrée un *tableau A* et un *indice i* dans le tableau.

Quand *ENTASSER* est appelée, on suppose que les arbres binaires enracinés en dans les fonctions *gauche(i)* et *droit(i)* (vu précédemment) sont des tas mais *A[i]* peut être plus petit que ses fils, violant ainsi *la propriété de tas.*

Le rôle de *ENTASSER* est de faire *« descendre »* la valeur *A[i]* dans le tas de manière que le sous arbre enraciné en *i* deviennent *un tas.*

Procédure ENTASSER ({E}A, i : entier)

var

lnd ,r , max, permut : entier

Début

lnd ← gauche(i) *//********** la fonction gauche ramène l'indice du fils gauche*

r ← droit(i) *//********** la fonction droit ramène l'indice du fils droit*

si (lnd <= taille(A) et A[lnd]>A[i]) alors *//*** vérifie si la valeur de l'indice du fils gauche n'est pas en dehors du tas et aussi si la valeur du tableau se trouvant à l'indice du fils gauche est supérieure à la valeur du parent*

max ← lnd *//**** Si les deux conditions du si sont vérifiées alors max reçoit la valeur de l'indice gauche*

sinon

max ← i *//**** Si les deux conditions du si ne sont pas vérifiées alors max reçoit la valeur de l'indice du pere*

finsi

si (r <= taille(A) et A[r] > A[max]) alors *//*** vérifie si la valeur de l'indice du fils droit n'est pas en dehors du tas et aussi si la valeur du tableau se trouvant à l'indice du droit est supérieure à la valeur se trouvant à l'indice max*

max ← r *//**** Si les deux conditions du si sont vérifiées alors max reçoit la valeur de l'indice droit*

finsi

si max <> i alors *//*******si l'indice max est différent de l'indice i alors on fait une permutation et la procédure est appelée de manière récursive*

permut ← A[i]

A[i] ← A[max]

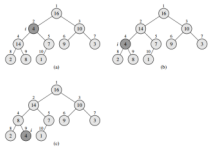
A[max] ← permut

ENTASSER (A, max)

fin si

fin

Illustration ci-dessous



2.2- Construction d'un tas

La procédure *ENTASSER* peut s'utiliser à l'envers pour convertir un tableau A [1..n] avec n = longueur (A)

En effet, les éléments A [(n mod 2)..n] sont des feuilles de l'arbre, chacun est au départ un tas à un élément.

La procédure *CONSTRUIRE_TAS* traverse les nœuds restants et exécute ENTASSER sur chacun d'eux.

L'ordre dans lequel les nœuds sont traités garanti que les sous arbres enracinés aux fils d'un nœud i sont des tas avant que ENTASSER soit exécutée à ce nœud.

Procédure CONSTRUIRE_TAS ({E}A : entier)

Début

Pour i ← (longueur (A) mod 2) à 1 faire

ENTASSER (A, i)

fin pour

fin

2.3- Algorithme Tri par tas

L'algorithme du tri par tas commence par utiliser CONSTRUIRE_TAS pour *construire un tas sur le tableau A [1..n] ou n = longueur (A).*

Comme l'élément maximal du tableau est stocké à la racine A[1], on peut le placer dans sa position finale correcte en l'échangeant avec A[n].

Si l'on « ôte » à présent le nœud n du tas (en décrémentant taille[A]), on observe que A[1 . . (n– 1)] peut facilement être transformé en tas max.

Les fils de la racine restent des tas, cependant la nouvelle racine peut violer la propriété des tas. Si c'est le cas, un seul appel, ENTASSER (A, 1), restaure la propriété de tas, qui laisse derrière un A [1... (n-1)].

L'algorithme du tri par tas répète alors ce processus pour le tas de taille n-1 jusqu'au tas de taille 2.

Procédure TRIER_TAS(A)

var

var

i,permut : entier

Début

CONSTRUIRE_TAS ({E}A : entier)

Pour i ← longueur(A) à 2 faire

permut ← A[1]

A[1] ← A[i]

A[i] ← permut

taille(A) ← taille(A) – 1

ENTASSER (A, 1)

fin pour

fin

Application 2 :

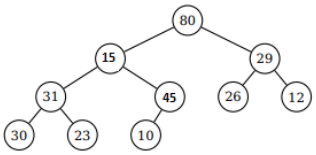
IV

Exercice

Un tri par tas commence par utiliser

- Une fonction pere
- Une procédure ENTASSER
- Une procédure CONSTRUIRE_TAS

Exercice



Soit le graphique ci-dessus, et considérons un tableau Tab.

L'appel qui permet de restaurer la propriété de tas est

Compléter la procédure de tri par tas de notre cas

Procédure TRIER_TAS(Tab)

var

i,permut : entier

Début

Pour ← à faire

permut ←

←

← permut

taille( ) ← taille( ) – 1

fin pour

fin