Analyse du tri par tas

Référence : Introduction à l'algorithmique, Cormen

Tas-max. la structure de tas max est une structure de données qu'on peut représenter par un arbre binaire complet gauche, où chaque étiquette d'un nœud est plus grande que celle de ses descendants. Un tas peut être implanter par tableau à n éléments numérotés de 1 à n, où le fils gauche (resp. droit) d'un élément $i \in \{0, ..., n-1\}$ est **GAUCHE**$(i) = 2i$ (resp. **DROITE**$(i) = 2i + 1$).

Tri par tas. L'idée du tri par tas est assez élémentaire une fois qu'on a la structure de donnée : étant donné les n éléments à trier, on construit un tas les contenant puis on extrait n dois le minimum du tas.

Algo ci-dessous à détailler, avec un exemple par exemple.

Algorithme 1 : Tasser(T,i,n)

```
g ← 2i ;
d ← 2i ;
# On veut récupérer le plus grand élément parmi T[i], T[g] et T[d]
si g ⩽ n et T[g] > T[d] alors
|  plus_grand ← g ;
sinon
└  plus_grand ← i ;
si d ⩽ n et T[d] > T[plus_grand] alors
└  plus_grand ← d ;
si plus_grand ≠ i alors
|  T[i], T[plus_grand] ← T[plus_grand], T[i] ;
└  Tasser(T, plus_grand,n) ;
```

Algorithme 2 : Construire(T,n)

```
pour i = ⌊n/2⌋, ..., 1 faire
└  Tasser(T,i,n) ;
```

Algorithme 3 : TriParTas(T,$i-1$)

```
Construire(T,n) ;
pour i = n − 1, ..., 1 faire
|  T[1], T[i] ← T[i], T[1] ;
└  Tasser(T,1,n − 1) ;
```

Analyse.

Lemme 1. *Soit $1 \leqslant i \leqslant n$, si les arbres enracinés en les fils de i sont des tas, alors après Tasser(T,i,n), l'arbre enraciné en i vérifie la propriété du tas-max, en temps $\mathcal{O}(h(i))$, où $h(i)$ est la hauteur de l'arbre enraciné en i et n le nombre total de nœuds.*

Démonstration. On note C_h la complexité dans le pire des cas de l'algorithme Tasser pour un nœud i dont l'arbre enraciné en i est de hauteur h. On a alors dans le pire cas $C_h = \mathcal{O}(1) + C_{h-1}$ et donc $C_h = \mathcal{O}(h)$.

Prouvons maintenant la correction. On montre par induction structurelle sur les arbres binaires que l'arbre obtenue vérifie la propriété du tas-max.

Si i est une feuille, alors Tasser(T,i,n) ne modifie pas T qui est déjà un tas.

Sinon, on fait une disjonction de cas selon la valeur de `plus_grand` :

— si `plus_grand`=i, alors l'arbre enraciné en i est un tas ;

— si `plus_grand`=g, alors on a si $d < n$ $T[g] \geqslant T[i]$ et $T[g] \geqslant T[d]$. On échange alors dans T les nœuds i et g et la propriété du tas-max est vérifiée localement. Puisque l'arbre enraciné en g était bien un tas par hypothèse, ses enfants le sont aussi et donc ont peut appliquer l'hypothèse d'induction sur g et après Tasser(T,g,n), l'arbre enraciné en g vérifie la propriété du tas-max.

— si `plus_grand`=d, on raisonne de la même manière.

□

Lemme 2. *L'algorithme Construire(T,n) transforme T en un tas en temps $\mathcal{O}(n)$.*

Démonstration.

Complexité : en notant $C(n)$ la complexité dans le pire des cas de l'appel Construire(T,n). Or, on sait qu'un tas à n éléments a une taille bornée par $\lfloor \log_2 n \rfloor$ et le nombre de nœud ayant la hauteur h est au plus $\lceil n/2^{h+1} \rceil$. Puisque notre fonction appelle la fonction `Tasser` pour tout nœud ayant une hauteur strictement positive, on a :

$$C(n) = \sum_{h=1}^{\lfloor \log_2 n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil \mathcal{O}(h) = \mathcal{O}\left(\sum_{h=1}^{\lfloor \log_2 n \rfloor} \frac{n}{2^h} h \right)$$

Or, on peut calculer la deuxième somme en dérivant la série géométrique de raison x et en appliquant la formule en 1/2. On obtient alors

$$\sum_{h \geqslant 0} \frac{h}{2^h} = \frac{1/2}{(1-1/2)^2} = 2$$

et ainsi $C(n) = \mathcal{O}(n)$.

Correction : Le fait que l'arbre obtenue soit complet gauche est direct via la représentation par tableau. Il reste à montrer qu'il vérifie la propriété du tas-max. On utilise l'invariant de boucle suivant : « Au début de chaque itération de la boucle, chaque nœud $i+1$, $i+2$, ... , n est la racine d'un arbre vérifiant la propriété du tas max.

— **Initialisation :** Avant la première itération, $i = \lfloor n/2 \rfloor$ et pour $j > \lfloor n/2 \rfloor$, alors l'arbre enraciné en j contient une unique feuille, et vérifie donc la propriété du tas-max trivialement.

— **Hérédité :** Pour voir que chaque itération conserve l'invariant, observez que les enfants du nœud i ont des numéros supérieurs à i. D'après l'invariant, ce sont donc tous les deux des racines d'un arbre vérifiant la propriété du tas-max. D'après le lemme 1, après `Tasser`(T,i,n), l'arbre enraciné en i vérifie la propriété du tas-max. Ainsi, tous les arbres enracinés en i, $i+1$,etc vérifie la propriété du tas-max.

À la fin, on a $i = 0$ et donc l'arbre enraciné en 1, c'est-à-dire T, vérifie la propriété du tas-max. □

Théorème 1. *La fonction* `TriParTas`*(T, n) trie en place un tableau T à n éléments en temps* $\mathcal{O}(n \log n)$.

Démonstration.

Complexité : D'après le lemme 2, l'appel `Construire(`T`,`n`)` transforme T en un tas en temps $\mathcal{O}(\log n)$ sans prendre de place supplémentaire. De plus, l'algorithme fait $n - 1$ appels échanges et appels à la fonction `Tasser` qui s'exécute dans le pire des cas en $\mathcal{O}(\log n)$. Donc l'algorithme s'exécute en $\mathcal{O}(n + n \log n) = \mathcal{O}(n \log n)$.

Correction : On pose l'invariant suivant : « à chaque itération, le sous-tableau $T[i + 1...n]$ contient les $n - i - 1$ plus grands éléments de T triées, et $T[1...i]$ est un tas max ».

— Avant de rentrer dans la boucle, on a $i = n$ et donc $T[n + 1, n]$ est vide et $T[1...n]$ est effectivement un tas d'après le lemme 2.

— On suppose que pour $i > 0$, on a $T[i + 1...n]$ trié et qui contient les plus grands éléments et $T[1...i]$ est un tas. On a alors $T[1]$ le plus grand élément de $T[1...i]$, et donc après échange, $T[i...n]$ est encore trié et contient les $n - i$ plus grands éléments de T. Ensuite, d'après le lemme 1, après l'appel `Tasser`(T,0, $i - 1$), on a $T[1...i - 1]$ qui est un tas.

À la fin de la boucle, on a alors $i = 0$ et donc $T[1...n]$ qui est trié. □