Analyse du Tri Par Tas

Julien Devevey

2018-2019

Ref : Cormen - Introduction to Algorithms p. 150 à 160

Remarque 1. La structure de tas max est une structure de données qu'on peut représenter par un arbre binaire vérifiant la définition par récurrence suivante : les feuilles sont des tas, et un arbre enraciné en i est un tas si les sous-arbres enracinés en les deux fils de i sont des tas, qu'ils ont la même hauteur et que l'étiquette de i est plus grande que celle de ses fils. Or il est possible de représenter un arbre en pratique par un tableau, où l'on implémente les fils gauche et droit par GAUCHE$(i) = 2i$ et DROIT$(i) = 2i + 1$.

Algo 2. Tri Par Tas

Algorithm 1 Tasser

Entrée: (T, i, n)

1: $g \leftarrow$GAUCHE(i)
2: $d \leftarrow$DROIT(i)
3: **si** $g < n$ et $T[g] > T[d]$ **alors**
4: plus_grand$\leftarrow g$
5: **sinon**
6: plus_grand$\leftarrow i$
7: **fin si**
8: **si** $d < n$ et $T[d] > T[$plus_grand$]$ **alors**
9: plus_grand$\leftarrow d$
10: **fin si**
11: **si** plus_grand$\neq i$ **alors**
12: Echanger $T[i]$ avec $T[$plus_grand$]$
13: Tasser(T,plus_grand$, n$)
14: **fin si**

Algorithm 2 Construire_Tas_Max

Entrée: (T, n)

1: **pour** $i = \lfloor n/2 \rfloor - 1$ à 0 **faire**
2: Tasser(T, i, n)
3: **fin pour**

Algorithm 3 Tri_Par_Tas

Entrée: (T, n)

1: Construire_Tas_Max(T, n)
2: **pour** $i = n - 1$ à 1 **faire**
3: Echanger $T[0]$ avec $T[n-1]$
4: Tasser$(T, 0, i-1)$
5: **fin pour**

Lemme 3. Si les arbres enracinés en les fils de i sont des tas, Tasser(T, i, n) transforme l'arbre enraciné en i en tas, en temps $O(\log(n) - h(i))$, où $h(i)$ est la hauteur de i dans l'arbre et n le nombre de noeuds de l'arbre.

Démonstration.

Soit $h(i)$ la hauteur du noeud dans l'arbre sur lequel on appelle Tasser. Alors en appelant c_h la complexité de l'algorithme pour un noeud de hauteur h dans le pire cas, on a $c_h = O(1) + c_{h+1}$ et on s'arrête uniquement quand on atteint une feuille, de profondeur environ $\log(n)$, comme on a des tas. D'où la complexité de l'algorithme en $O(\log(n) - h(i))$.

Prouvons sa correction par induction structurelle. Si i est une feuille, l'arbre enraciné en i est toujours un tas. On obtient donc bien un tas après un appel à Tasser. Si maintenant i est un noeud, et les arbres enracinés en ses fils sont des tas, et un appel à Tasser sur des noeuds plus bas dans l'arbre crée bien des tas si les arbres enracinés en leurs fils sont des tas. Alors, les arbres enracinés en les fils des fils de i sont des tas au début de l'algorithme par hypothèse, et comme on échange des valeurs uniquement entre i et ses fils avant l'appel récursif, c'est toujours vrai au moment dudit appel, donc l'appel récursif transforme l'arbre enraciné en GAUCHE(i) ou DROITE(i) (suivant le cas) en un tas, et l'arbre enraciné en l'autre fils était un tas, et n'a pas été modifié, donc est toujours un tas.

On a aussi que $T[i] \geq \max(T[\text{GAUCHE}(i)], T[\text{DROITE}(i)])$.

Donc l'arbre enraciné en i vérifie la définition de tas. □

Lemme 4. Construire_Tas_Max(T, i, n) transforme l'arbre enraciné en i en un tas, en considérant le tableau T comme un arbre, en un temps $O(n)$.

Démonstration.

Regardons la complexité de l'algorithme. Il effectue $\lfloor n/2 \rfloor$ appels à Tasser, il tourne donc en un temps $O(n \sum_{h=0}^{\lfloor \log(n) \rfloor} \frac{h}{2^h})$, car il y a $\lceil n/2^{n+1} \rceil$ noeuds de

hauteur h dans un arbre équilibré, et qu'on change d'indice en remplaçant h par $\lfloor \log(n) \rfloor - h$. On reconnaît alors la somme partielle de la dérivée de la série géométrique de raison $1/2$, ce qui permet de dire que la complexité de l'algorithme est $O(n)$.

Montrons ensuite qu'il est correct par récurrence sur i, en posant un invariant de boucle (H_i) "Les arbres enracinés en $k > i$ sont des tas au début de l'itération i.".

- **Initialisation** : Pour $i \geq \lfloor n/2 \rfloor$, i représente une feuille, c'est donc bien toujours un tas et l'invariant est vrai avant qu'on entre dans la boucle.
- **Hérédité** : Soit $i < \lfloor n/2 \rfloor$ et supposons que $\forall k > i$, l'arbre enraciné en k soit un tas. D'après le lemme précédent, les arbres enracinés en les fils de i sont des tas, et alors l'appel à Tasser transforme l'arbre enraciné en i en un tas. De plus, si $k > i$ n'est pas un descendant de i, il n'est pas modifié lors de l'appel à Tasser, et l'arbre enraciné en k est encore un tas. Si c'est un descendant de i, comme l'arbre enraciné en i est un tas, on a par définition que l'arbre enraciné en k est un tas.

Ainsi, à la fin de la boucle l'invariant est toujours vrai et en particulier, l'arbre enraciné en 0 est un arbre : l'arbre entier est donc bien un tas. □

Théorème 5. Tri_Par_Tas(T, n) trie en place un tableau T de taille n en un temps $O(n \log(n))$.

Démonstration.

L'algorithme effectue $n-1$ appels à Tasser et autant d'échanges d'éléments. Il a alors une complexité dans le pire cas de $O(n+(n-1)(\log(n)+1)) = O(n \log(n))$. Après l'appel à Construire_Tas_Max, T est un tas. Donc $T[0]$ contient le plus grand élément du tableau. On pose alors l'invariant de boucle suivant (P_i) "Le sous-tableau $T[i+1..n-1]$ contient au début de l'étape i les $n-i-1$ plus grands éléments de T, triés et l'arbre $T[0..i]$ est un tas."

- **Initialisation** : Avant la première itération, pour $i = n-1$, le sous-tableau considéré est vide et le lemme 3 assure que T est un tas, et l'invariant est donc vrai.
- **Hérédité** : Si au début de la i-ème itération (P_i) est vraie, alors $T[0]$ contient le plus grand élément de $T[0..i]$, qui est aussi plus petit que tous les éléments de $T[i+1..n-1]$. Alors après l'échange, $T[i..n-1]$ contient bien les $n-i$ plus grands éléments de T triés. De plus, les sous-arbres enracinés en 1 et 2 sont encore des tas, donc l'appel à Tasser transforme bien $T[0..i-1]$ en arbre. Alors, à la fin de la i-ème itération, (P_{i-1}) est vraie.

Conclusion : à la fin de la boucle, (P_0) est vraie, c'est à dire que le tableau T est trié. □