Tri par tas

Léo Gayral

2017-2018

ref : Cormen – Introduction to algorithms, third edition – p.154

Définition 1. Pour un arbre T, on définit $h(T) = \max_{x \in T} h(x)$ sa hauteur, avec la convention $h(x) = 1$ pour la racine.

On considère la structure de *tas-max*, qui désigne des arbres binaires doublement chaînés, dont la racine de chaque sous-arbre est le maximum de tous ses nœuds. De façon équivalente, tout chemin de hauteur croissante – appelé une *branche* par la suite – est ordonné par ordre décroissant.

On peut représenter un tas-max de profondeur h dans un tableau T de taille $2^h - 1$, de sorte que le fils gauche de $T[i]$ (resp. droit) est $T[2i+1]$ (resp. $T[2i+2]$), quitte à remplir les cases vides par des $-\infty$ pour préserver la propriété de décroissance le long des branches.

Proposition 1. La structure tas-max est munie des opérations suivantes :
- $INSERER(x, T)$, qui ajoute l'élément x aux noeuds de T en $O(h)$,
- $EXTRAIRE(T)$, qui retourne le maximum de T en $O(h)$,
- $FUSION\,(T_1, T_2)$, en $O\left(\max\left[h\left(T_1\right), h\left(T_2\right)\right]\right)$.

Démonstration.
L'insertion de x dans T est un cas particulier de percolation ascendante.

Pour cela, on choisit un noeud de T avec au plus 1 descendant – ce qui peut se faire en $O(h)$ – auquel on ajoute x sur un des descendants vacants. Idéalement, ce nœud est de profondeur minimale pour éviter de déséquilibrer T, mais ce n'est pas nécessaire pour préserver sa structure.

Ensuite, on fait remonter x au sein l'arbre tant que x n'est pas la racine et est supérieur à son père. Ce-faisant, on préserver la propriété « le sous-arbre enraciné en x est un tas-max », et lorsque l'algorithme s'arrête, soit x est la

racine de T, soit x est inférieur ou égal à son père. Dans les deux cas, T n'a que des branches décroissantes, donc est un tas-max.

En outre, comme x part d'une feuille, on peut au plus remonter h fois au sein de l'arbre avant d'atteindre la racine auquel cas l'algorithme termine, d'où la complexité annoncée.

Pour l'extraction de la racine, on suit de même un principe de percolation descendante. Une fois le maximum mise de côté, on comble ce *vide* par permutation avec le maximum entre la racine du fils gauche et celle du fils droit, ce qui place un vide à la racine du sous-arbre en question. On itère ce procédé jusqu'à créer un vide à la racine d'un arbre qui a au plus un de ses deux fils non vide, auquel cas on place ce fils directement à la racine. Comme précédemment, on déplace ce vide le long d'une branche, d'où au plus h itérations.

Ce-faisant, on préserve la propriété « toute sous-branche ne contenant pas ce *vide* est décroissante », ce qui signifie que l'arbre final, duquel on a éjecté le vide, est bien un tas-max. □

Proposition 2. Étant donné une liste T à n éléments, on peut en particulier créer un tas-max en place, en $O(n)$ opérations.

Démonstration.
La méthode naïve de construction d'un tas consiste à insérer un à un les éléments en partant du tas vide, ce qui donne une complexité en $O(n \ln(n))$.

On peut cependant considérer la liste comme un arbre binaire avec la même structure que pour le tas. Initialement, chacune des feuilles de cet arbre est bien un tas-max. En partant de la fin de la liste, par hauteur décroissante, on effectue une percolation descendante en l'arbre enraciné en $T[i]$, comme pour $EXTRAIRE(T)$ mais en faisant remonter la racine dans ce sous-arbre au lieu du vide (qui vaut *moralement* $-\infty$), ce qui donne un tas-max enraciné en $T[i]$. On obtient finalement un tas-max enraciné en $T[0]$, donc une structure de tas-max sur T.

Cet arbre a (au plus) 2^i noeuds à la hauteur $i \leq h \sim log_2(|T|)$, chacun demandant une percolation en $O(h-i)$ étapes, donc la complexité globale de ce algorithme est en :

$$C(n) = \sum_{i=1}^{h} 2^i \times O(h-i) = 2^h \sum_{j=1}^{h} \frac{O(j)}{2^j}$$

or la série $\left(\frac{j}{2^j}\right)_{j\in\mathbb{N}}$ est sommable, d'où $C(n) = O(n)$. □

Application 1. L'algorithme de tri par tas permet de trier une liste en place, en $O(n\ln(n))$:

```
1  def HEAPSORT(L) :
2      CREER_TAS(L)
3      pour i de |L| − 1 a 1 :
4          m = EXTRAIRE(T[: i])
5          T[i] = m
```

Démonstration.
Pour que cet algorithme fonctionne tel quel, il faut cependant apporter quelques modifications à la procédure d'extraction, initialement conçue pour des arbres quelconques.

Ici, on connait la position des feuilles, donc on peut directement mettre la feuille la plus à droite, en position i dans T, pour la mettre à la racine et effectuer une percolation ascendante au sein du tas $T[: (i-1)]$, de sorte qu'on puisse ensuite ajouter m en position i sans écraser une entrée de l'arbre. □